INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

CONTROL BASADO EN SIP DE VÍDEO SOBRE IP EN TELÉFONOS MÓVILES

Autor: Víctor Álvarez Vallejo
Tutor: Carlos García Rubio

Leganés, 27 de octubre de 2010

Agradecimientos

A Elena, por tu amor y tu apoyo incondicional. Por tu ayuda e infinita paciencia. Gracias por llenar mi vida de felicidad.

A mis padres, por su apoyo y su cariño, por darme aliento en los momentos de flaqueza. A mis tías, por su cariño y por animarme a superar las adversidades; y a mi tío, porque sé que este momento le hubiera colmado de alegría, y deseo que estas palabras le lleguen donde quiera que esté. Sin vosotros no habría conseguido llegar hasta aquí.

A mi tutor Carlos, por su inestimable ayuda. Gracias por guiarme a cada paso durante todo este tiempo.

A mis compañeros del Departamento de Ingeniería Telemática, por vuestros consejos y ayuda. Gracias por los buenos ratos que me habéis hecho pasar en el laboratorio 4.1.A01.

Resumen

Según múltiples estudios de mercado, en la próxima década el acceso móvil a Internet igualará o superará al acceso desde los ordenadores de escritorio, por lo que el desarrollo de aplicaciones móviles representa un sector con nuevas e interesantes oportunidades de negocio.

Esta situación se debe fundamentalmente a dos factores. Actualmente, los celulares dispone de una potente y nueva tecnología que permite el planteamiento de nuevas aplicaciones hasta ahora impensables para los desarrolladores, y que abren un nuevo camino donde la imaginación es el único límite. Por otro lado, los nuevos planes de datos, de diversas compañías de telefonía móvil, permiten al usuario estar constantemente conectado a la red, al igual que si lo estuviera desde su casa con el ya omnipresente ADSL.

Esta coyuntura, hace que nos encontremos ante un momento crítico para la movilidad en Internet, donde la filosofía *always-on* permitirá diferenciarse de la competencia, facilitando la gestión y el descubrimiento de nuevas oportunidades de negocio.

En este contexto, la plataforma S60 representa una de las interfaces móviles más extendidas en el mundo a día de hoy, contando con millones de usuarios en todo el planeta. Desarrollada principalmente por Nokia, uno de los primeros fabricantes de teléfonos móviles, así como otras empresas como Lenovo, LG Electronics, Panasonic o Samsung; la serie S60 representa un estándar multi-venta de teléfonos móviles que permite la instalación de nuevas aplicaciones tras la adquisición del dispositivo, soportando software desarrollado en Java, C++ o Python.

Este proyecto pretende aprovechar las capacidades ofrecidas por la serie 60 de terminales móviles, para la implementación de aplicaciones multimedia que permitan la movilidad en Internet en dispositivos Symbian S60. Para ello se utilizará el lenguaje de programación Python para S60, denominado PyS60, el cual constituye un conjunto de módulos o librerías basadas en extensiones Symbian C++, enfocadas al desarrollo de software para dispositivos S60.

En este sentido, en el presente trabajo se implementará un sistema para el envío de flujos M-JPEG sobre RTP, desde un terminal móvil a un equipo fijo, que permita la movilidad en Internet mediante el protocolo SIP, lo que involucrará un estudio paralelo de las capacidades ofrecidas por PyS60 para la creación de aplicaciones móviles para terminales S60.

Abstract

According to several market research, Internet mobile access will match (or exceed) Internet desktop access in the next decade, so mobile applications devolopment represents a sector with new and exciting business opportunities.

This situation is due to two factors. Currently, mobile phones have got new technology that it allows the approach of new unimaginable applications, and it opens a new path where the imagination is the limit. On the other hand, new data plans of several mobile companies, they are allowing that the user can be online all the time, as when he is at home with ADSL service.

This situation makes that we are in a critical moment for Internet mobility, where the "always-on" philosophy will allow to be different from competition, providing management and discovery of new business opportunities.

In this context, S60 mobile interface is one of the most widespread in the world today, with millions of users. S60 platform is primarily developed by Nokia, one of the first mobile manufacturers, and other companies such as Lenovo, LG Electronics, Panasonic and Samsung. S60 is a multi-standard mobile phone platform that it allows to install new applications after device acquisition, supporting software developed in Java, C or Python.

This project wants to take advantage of S60 capabilities, for implementing Internet media applications for Symbian S60 phones. For this task, we will use Python programming languaje for S60, called PyS60, which is a modules set or libraries based on Symbian C++ extensions and it is focused on developing software for S60 devices.

In that sense, this study will implement a system for sending M-JPEG streams over RTP from a mobile to a fixed equipment. The system must support mobility for multimedia with SIP protocol. This document will involve a parallel study of the PyS60 capabilities for creating mobile applications for S60 terminals.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Durante el siglo pasado, el teléfono tradicional ha soportado las comunicaciones de voz durante generaciones, mientras que Internet ha desencadenado una revolución en el campo de la información. Pero ninguna de estas tecnologías ha progresado tan rápido como lo están haciendo los dispositivos móviles actuales, los cuales incorporan constantemente nuevas funcionalidades y capacidades.

La comunicación de voz y datos ahora se integra con funciones de valor agregado y aplicaciones que aumentan la productividad, en un dispositivo compacto que también funciona como una potente computadora.

En los últimos años estamos asistiendo a un despegue sin precedentes del cómputo y las comunicaciones móviles. Esta gran aceptación a nivel mundial se deriva de la integración de dichas tecnologías en nuestro trabajo y en el hogar, llegando a convertirse en un estilo de vida impensable para la mayoría hace años.

En estos momentos nos dirigimos hacia una situación que bien podría calificarse como "revolución móvil", que se basa en la idea de que todo los accesos que hoy se encuentran cableados se están convirtiendo en accesos inalámbricos. *Hot spots* en hoteles, cafeterías, aeropuertos o universidades, al igual que las redes inalámbricas metropolitanas, son buenos ejemplos de ello. De hecho, podemos afirmar que estamos asistiendo a una etapa de popularización de la conectividad a la redes inalámbricas, las cuales permiten a los usuarios estar constantemente informados en cualquier momento mediante un terminal móvil.

La principal característica de los terminales móviles es que permiten comunicarse desde casi cualquier lugar, y aunque su principal función es la comunicación de voz, su rápido desarrollo ha incorporado otras funcionalidades como: cámara fotográfica, agenda, acceso a Internet, *Bluetooth*, infrarrojos, reproducción de vídeo y mp3, e incluso GPS, entre otros.

La convergencia del cómputo y las comunicaciones, ligadas a las características inherentes de movilidad y ubicuidad de los dispositivos móviles, está promoviendo un cambio en el paradigma del acceso a la información. De este modo, un usuario móvil podría establecer una sesión multimedia con otro dispositivo, para el intercambio de información desde un lugar con cobertura 3G, para posteriormente desplazarse a otra ubicación con cobertura Wi-Fi.

En este contexto, este proyecto pretende crear un sistema para el envío de vídeo, desde un terminal móvil, que ofrezca soporte para la movilidad en Internet mediante el protocolo SIP (*Session Initiation Protocol*), de tal forma que, ejemplos como el anterior, resulten viables.

Concretamente, la idea principal de este trabajo consiste en dar soporte de movilidad a un usuario, que previamente ha establecido una sesión multimedia con otro equipo para el envío de vídeo desde su terminal celular, de tal modo que, si el usuario móvil cambia de red, y por tanto su dispositivo adquiere una nueva dirección IP, la comunicación establecida pueda continuar de forma ininterrumpida y de modo transparente entre ambos extremos.

En primera instancia, se ha decidido que el vídeo enviado por el terminal móvil posea una codificación sencilla como M-JPEG, ya que el objetivo principal de este proyecto no reside en el formato de los datos transportados, si no el soporte SIP necesario que permita la movilidad del terminal móvil entre una red y otra.

Además, para simplificar el diseño, se ha decidido que el dispositivo receptor de dicho vídeo, consistirá en un equipo IP fijo convencional. De este modo, el planteamiento inicial solo involucrará dos tipos de terminales:

1. Terminal móvil, encargado de enviar un vídeo M-JPEG. Es capaz de cambiar de red, y por tanto, adquirir una nueva dirección IP.

2. Equipo receptor, máquina IP fija encargada de reproducir el vídeo M-JPEG enviado desde el terminal móvil. No cambia en ningún momento de red, por lo que posee siempre la misma dirección IP.

En lo que respecta al terminal móvil, el presente proyecto pretende aprovechar las capacidades ofrecidas por la serie 60 de dispositivos móviles, para la implementación de aplicaciones multimedia que permitan la movilidad en Internet mediante el protocolo SIP en dispositivos Symbian S60, tal y como se ha descrito anteriormente. Para ello se utilizará el lenguaje de programación Python para S60, denominado PyS60, el cual constituye un conjunto de módulos o librerías basadas en extensiones Symbian C++, enfocadas al desarrollo de software para dispositivos S60.

A partir de un primer estudio de las capacidades proporcionadas por dicha librería de módulos, se procederá a implementar un caso práctico de aplicación multimedia que permita la movilidad en Internet, realizando posteriormente una batería de pruebas en un entorno simulador móvil S60 que permita verificar su correcto funcionamiento.

En este sentido, este proyecto planteará una posible implementación de un sistema para el envío de flujos M-JPEG sobre RTP, desde un terminal móvil S60 a un equipo fijo, que permita la movilidad mediante los protocolos SIP y SDP, de tal forma que, si el terminal móvil cambia de red y adquiere una nueva dirección IP, la reproducción del flujo M-JPEG no se vea interrumpida en el equipo receptor.

Por otro lado, y de forma paralela, se generará un estudio sobre de las capacidades ofrecidas por PyS60 para la creación de aplicaciones móviles para terminales S60.

Los sucesivos capítulos del presente proyecto versan acerca de los detalles técnicos del sistema final implementado, así como las diversas cuestiones analizadas y los aspectos contemplados, a la hora de elegir la tecnología utilizada para su desarrollo.

En los siguientes epígrafes se ofrecerán las razones que motivan el presente trabajo, así como los objetivos que desean ser alcanzados a la finalización del mismo.

1.1. Motivación del proyecto

El desarrollo de aplicaciones móviles conlleva una variedad de consideraciones de acuerdo con el escenario y el propósito para el que van a ser utilizadas. En este aspecto, el paradigma descrito en la introducción (Capítulo 1) del presente estudio, motiva algunas de las razones de este proyecto:

- Implementación de un sistema final que permita la movilidad en Internet mediante el protocolo SIP, cuyo desarrollo esté basado en el lenguaje de programación Python y el conjunto de librerías PyS60 para terminales S60.
- Implementación de una aplicación multimedia para terminales Symbian S60 basada en PyS60 que permita el envío de un flujo de imágenes capturadas, o almacenadas previamente en el terminal, a otro equipo fijo.
- Implementación de una aplicación multimedia para terminales Symbian S60

basada en PyS60 que permita la movilidad de usuarios entre diferentes redes.

- Estudio de un lenguaje de programación, así como un conjunto de librerías, que permitan la implementación de aplicaciones móviles para un conjunto amplio de dispositivos.

- Estudio de las capacidades ofrecidas por el lenguaje de programación Python, así como el conjunto de librerías PyS60, para el acceso rápido y sencillo a las capacidades multimedia de los terminales Symbian de la serie 60: cámara fotográfica, establecimiento de una conexión GPRS, información de localización GPS, etc.; así como el manejo de *widgets* o elementos nativos para la interfaz gráfica de usuario o GUI de la plataforma S60.

- Verificación de que PyS60 proporciona la capacidad de desarrollar, de forma rápida y eficaz, aplicaciones para una amplia gama de dispositivos.

Estas motivaciones guiarán los siguientes epígrafes del presente trabajo, a lo largo de los cuales analizaremos el potencial de PyS60 en el desarrollo de aplicaciones para el envío de información multimedia desde terminales móviles, la facilidad para implementar software de comunicaciones para la plataforma Symbian S60 mediante esta tecnología, así como el desarrollo de aplicaciones multimedia que permitan la movilidad de los usuarios entre distintas redes gracias al protocolo SIP.

1.2. Objetivos

El presente proyecto persigue un objetivo fundamental:

- Implementar un sistema que ofrezca soporte para la movilidad en Internet, mediante el protocolo SIP, a un terminal móvil que previamente haya establecido una sesión multimedia, para el envío de un vídeo o flujo M-JPEG, con un equipo IP receptor fijo.

Por otra parte, podemos enumerar algunos objetivos secundarios diferenciados:

1. Desarrollar un sistema que posea dos tipos de usuarios claramente diferenciados. Por una lado, los usuarios móviles, capaces de enviar un flujo de información multimedia desde su terminal. Por otra parte, los usuarios

fijos, capaces de reproducir la información en destino para su correcta visualización en el equipo receptor.

2. Desarrollar un sistema que permita el envío de vídeo o de una secuencia de imágenes desde un terminal móvil a un equipo receptor. En este sentido se propone el uso de RTP como protocolo de comunicaciones para transportar dicho flujo multimedia.

3. Utilizar una codificación sencilla para el envío del flujo de información multimedia desde el terminal móvil al equipo fijo. En este sentido se propone el uso una secuencia de imágenes M-JPEG u otro formato simple en su defecto.

4. Desarrollar un sistema que permita la movilidad de los usuarios móviles entre diferentes redes. Para soportar esta característica se utilizará el protocolo SIP. Así mismo, la aplicación móvil implementada debe ser capaz de gestionar la señalización y envío de información multimedia desde un terminal móvil S60, mientras que el equipo receptor debe ser capaz de reproducir la información recibida y gestionar la señalización SIP que permita soportar la movilidad requerida por el sistema.

5. Desarrollar un sistema que permita simular y gestionar un *hand-off* o cambio de red de un terminal móvil, en cualquier instante de la comunicación con el equipo receptor.

6. Estudiar y seleccionar un lenguaje de programación, así como un conjunto de librerías, que permitan la implementación de aplicaciones móviles para un conjunto amplio de dispositivos de forma rápida y eficaz. En este sentido, se decide optar por desarrollar software para la plataforma o conjunto de terminales móviles Symbian S60, debido al significativo número de unidades vendidas en todo el mundo actualmente. Se decide escoger Python como lenguaje de programación, y más concretamente PyS60, un conjunto de librerías específico para su uso en el desarrollo de aplicaciones para estos dispositivos.

7. Estudiar la situación actual de la implementación de las librerías Python para terminales S60, analizando sus capacidades y sus deficiencias. Verificar que es posible desarrollar de forma rápida y eficiente aplicaciones multimedia para terminales Symbian S60 mediante PyS60.

8. Desarrollar un sistema, basado en este lenguaje y conjunto de librerías, que permita el envío, recepción y señalización de un flujo de información multimedia entre un terminal móvil S60 y un equipo IP fijo receptor.

9. Probar la aplicación móvil desarrollada y verificar su correcto funcionamiento en el entorno simulador proporcionado por Nokia en el paquete software o SDK *S60 Developers Tools* para terminales de 3ª Edición FP1 (*Feature Pack 1*).

10. Desarrollar las funciones o módulos Python necesarios para la creación e interpretación de un grupo acotado de mensajes SIP.

11. Desarrollar las funciones o módulos Python necesarios para la creación e interpretación de un grupo acotado de mensajes SDP.

12. Desarrollar las funciones o módulos Python necesarios para la creación e interpretación de paquetes RTP.

13. Desarrollar las funciones o módulos Python necesarios para la creación e interpretación de paquetes M-JPEG sobre RTP.

1.3. Contenido de la memoria

El resto de la presente memoria se estructura como sigue:

- En el Capítulo 2 se introducen las diferentes tecnologías que constituyen el contexto del proyecto. En este sentido, se describe la plataforma de terminales móviles Symbian S60, a la que está enfocada el presente trabajo, los lenguajes de programación y los protocolos de comunicaciones utilizados para el desarrollo del mismo, las bases de datos SQLite, así como una breve descripción de la arquitectura para la movilidad de terminales multimedia basada en el protocolo SIP.

- En el Capítulo 3 se introducen los principales aspectos de los servicios básicos que deberá proporcionar el sistema implementado, una descripción general del funcionamiento del sistema, así como los requisitos funcionales y técnicos de las aplicaciones desarrolladas.

- El Capítulo 4 profundiza en los detalles de la arquitectura y el software implementado para el terminal móvil mediante el lenguaje de programación Python y el conjunto de librerías PyS60, especialmente desarrolladas para su uso en terminales Symbian S60.

- El Capítulo 5 profundiza en los detalles de la arquitectura y el software implementado para el equipo receptor mediante el lenguaje de programación

Python, el conjunto de librerías estándar de dicho lenguaje y la librería Python externa PIL.

- En el Capítulo 6 se describen los diversos módulos comunes implementados, los cuales dan soporte a las comunicaciones establecidas entre el terminal móvil y el equipo receptor.
- En el Capítulo 7 se presenta la batería de pruebas realizada al sistema completo, así como los subsistemas que lo componen.
- En el Capítulo 8 se describen las distintas fases de desarrollo del proyecto, explicando las tareas realizadas en cada una, los problemas encontrados y los resultados obtenidos.
- En el Capítulo 9 se resumen las principales conclusiones del proyecto realizado y se enumeran una serie de posibles líneas de trabajo futuro.

Como parte complementaria de la memoria, se ha incluido una serie de apéndices que intentan aclarar y completar algunos aspectos relacionados con la solución propuesta en este proyecto, así como posibles trabajos futuros que se apoyen en el presente estudio.

- En el Apéndice A se detalla el presupuesto del presente proyecto.
- En el Apéndice B encontramos una descripción de los pasos necesarios para llevar a cabo la instalación de las distintas herramientas y librerías utilizadas para el desarrollo de este proyecto. En este apéndice, también se incluyen los pasos necesarios para instalar el sistema propuesto, ya sea la aplicación móvil en un terminal S60 o un emulador software, o la aplicación reproductora en el equipo IP receptor.
- En el Apéndice C se explica como iniciar, configurar y ejecutar la aplicación práctica desarrollada durante este trabajo.
- En el Apéndice D se incluye diversa información introductoria para la creación y compilación de una extensión Symbian C++ para PyS60.
- En el Apéndice E se incluye un glosario de términos que aparecen a lo largo de esta memoria.

Capítulo 2

Estado del arte

A lo largo de este capítulo vamos a introducir las diferentes tecnologías involucradas en el presente proyecto. El objetivo principal consiste en proporcionar una visión general del estado actual de la plataforma Symbian S60, el desarrollo de aplicaciones mediante el lenguaje de programación Python y el conjunto de librerías PyS60 para terminales de la serie 60, así como los protocolos de comunicaciones y otros elementos utilizados en este trabajo.

2.1. Plataforma Symbian S60

El sistema desarrollado en el presente proyecto está basado en la plataforma Symbian S60 [1], ya que la aplicación implementada en la parte móvil está diseñada para su uso en terminales Symbian de la serie 60.

Symbian [2] es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia [3], Sony Ericsson, PSION, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus orígenes provienen de su antepasado `EPOC32`, utilizado en PDA's y *Handhelds* de PSION, y fue creado con el objetivo de desarrollar un sistema operativo capaz de competir con otros sistemas como Palm OS.

La plataforma S60 (formalmente, interfaz de usuario de serie 60) es una plataforma para terminales móviles que utilicen el sistema operativo Symbian OS, y está desarrollada principalmente por Nokia, uno de los primeros fabricantes de teléfonos móviles del mundo y una de las principales empresas del sector de las telecomunicaciones, además de otros fabricantes que también poseen una licencia de uso, incluyendo a Lenovo, LG Electronics, Panasonic y Samsung.

Existe una amplia comunidad de desarrolladores que programa aplicaciones para esta plataforma, así como una serie de compañías que también lo hacen:

- Operadores de telefonía como Vodafone y Orange, que desarrollan y proveen aplicaciones y servicios basados en S60.
- Compañías de integración de software como Elektrobit, Teleca y Sysopen Digia.
- Compañías de semiconductores como Texas Instruments, STMicroelectronics, Broadcom, Renesas y Freescale.

Aunque en este estudio nos centraremos en el desarrollo de aplicaciones para terminales móviles de la plataforma S60, Symbian cuenta con varias interfaces de usuario o plataformas para su sistema operativo:

- Serie 60, Serie 80, Serie 90: Usadas por la mayoría de los móviles con Symbian y elegidas por la *Symbian Foundation* como plataformas a continuar.
- UIQ: Desarrollada por UIQ Technology, es usada principalmente por Sony-Ericsson y Motorola.
- MOAP: Usada por algunos móviles 3G de NTT-Docomo.

Concretamente, la plataforma S60 consiste en un conjunto de bibliotecas y aplicaciones informáticas estándar, tales como: telefonía, herramientas de gestión de información personal o reproductores multimedia. Está enfocada a todos aquellos terminales móviles que hagan uso del sistema operativo Symbian OS, y se encuentra actualmente entre las plataformas líderes de terminales móviles, representando actualmente una de las interfaces más extendidas, contando con millones de usuarios en todo el mundo.

Podemos definir el software S60 como un estándar multi-venta para terminales móviles, capaz de soportar aplicaciones desarrolladas mediante diversos lenguajes de programación como Java, C++ o Python. Una característica interesante que utilizaremos en este trabajo, ya que los terminales S60 permiten instalar nuevas aplicaciones tras su adquisición.

2.1.1. Modelos S60

Dentro de la plataforma S60, existen diferentes versiones desarrolladas a lo largo del tiempo. A continuación se muestra una tabla 2.1 donde se detallan los

modelos de los terminales [4] que incorporan cada una de las versiones de S60, así como la versión del sistema operativo Symbian en que se basan.

Edición de S60	Versión S60	Versión Symbian OS	Dispositivos
S60 versión 0.9	0.9	6.1	Nokia 7650
S60 1ª Edición	1.2	6.1	Nokia 3600, Nokia 3620, Nokia 3650, Nokia 3660, Nokia N-Gage, Nokia N-Gage QD, Sendo X, Sendo X2, Siemens SX1
S60 2ª Edición	2.0	7.0s	Lenovo P930, Nokia 6600, Panasonic X700, Panasonic X800, Samsung SGH-D720, Samsung SGH-D730
S60 2ª Edición, *Feature Pack* 1	2.1	7.0s	Nokia 3230, Nokia 6260, Nokia 6620, Nokia 6670, Nokia 7610
S60 2ª Edición, *Feature Pack* 2	2.6	8.0a	Nokia 6630, Nokia 6680, Nokia 6681, Nokia 6682
S60 2ª Edición, *Feature Pack* 2	2.8	8.1a	Nokia N70, Nokia N72, Nokia N90
S60 3ª Edición	3.0	9.1	Nokia 3250, Nokia 5500 sport, Nokia E50, Nokia E60, Nokia E61, Nokia E62, Nokia E65, Nokia E70, Nokia N71, Nokia N73, Nokia N75, Nokia N77, Nokia N80, Nokia N91, Nokia N91 8GB, Nokia N92, Nokia N93

S60 3ª Edición *Feature Pack* 1	3.1	9.2	LG KT610, Nokia 5700 XpressMusic, Nokia 6110 Navigator, Nokia 6120 Classic, Nokia 6121 Classic, Nokia 6124 Classic, Nokia 6290, Nokia E51, Nokia E66, Nokia E71, Nokia E90 Communicator, Nokia N76, Nokia N81, Nokia N81 8GB, Nokia N82, Nokia N95, Nokia N95 8GB, Samsung SGH-G810, Samsung SGH-L870
S60 3ª Edición *Feature Pack* 2	3.2	9.3	Nokia 5320 XpressMusic, Nokia 6210 Navigator, Nokia 6220 Classic, Nokia 6720 Classic, Nokia 6710 Classic, Nokia 6650, Nokia N78, Nokia N96, Nokia N79, Nokia N85, Nokia N86, Samsung GT-i8510, Samsung SGH-i450
S60 5ª Edición	3.3	9.4	Nokia 5800, XpressMusic Nokia N97

Cuadro 2.1: Lista de modelos de terminales S60

Puesto que el mundo de los terminales móviles está en constante evolución se recomienda visitar el sitio web oficial de S60, donde el lector podrá encontrar una lista detallada de los dispositivos S60 actuales.

`http://www.symbian.org/devices`

Cuadro 2.2: Lista actualizada de terminales S60

Cabe destacar que el software escrito para la primera edición (S60v1) y la segunda edición (S60v2) no es compatible a nivel binario con la tercera edición de la plataforma (S60v3), pues utiliza una versión nueva del sistema operativo Symbian OS (v9.1).

En nuestro caso particular, la aplicación práctica desarrollada en este trabajo se ha realizado bajo la plataforma S60 3ª Edición *Feature Pack* 1.

2.2. Lenguaje de programación utilizado

2.2.1. Python

El sistema implementado en el presente proyecto se ha desarrollado enteramente mediante el lenguaje de programación Python [5], tanto en la parte del terminal móvil, encargado de enviar el flujo de datos multimedia y simular los procedimientos de *hand-off* o cambio de red, así como en la parte del equipo receptor, encargado de reproducir la información recibida y gestionar las actualizaciones de sesión pertinentes mediante el protocolo SIP, que permitan soportar la funcionalidad de movilidad requerida por el sistema.

Python es un lenguaje de programación que puede considerarse como un lenguaje interpretado o de *script*, con tipado dinámico, fuertemente tipado, orientado a objetos y multiplataforma.

La idea de utilizar Python, para el desarrollo del software implementado en este proyecto, persigue el objetivo de explotar una serie de cualidades inherentes a dicho lenguaje:

- Es libre y de código abierto: Python es FLOSS (*Free and Open Source Software*), un concepto que se basa en la idea de una comunidad de desarrolladores que comparten sus conocimientos. Se pueden distribuir libremente copias de su software, leer su código fuente, aplicarle cambios, usar partes del mismo en nuevos programas libres, y en general cualquier modificación que se nos ocurra.

- Portable: Debido a su naturaleza de código abierto, Python ha sido portado a diversas plataformas, estando disponible para entornos: Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE y PocketPC.

- Lenguaje de alto nivel: Python nos permite concentrarnos en el problema que queremos resolver obviando los detalles de bajo nivel, como por ejemplo, gestionar la memoria dinámica empleada por nuestra aplicación. En este aspecto, Python pertmite la programación orientada a objetos, así como un estilo orientado a procedimientos, donde el programa se construye a partir de funciones o partes reutilizables de código.

- Es simple y sencillo de aprender: Python es en lenguaje minimalista. El pseudo-código natural de Python es una de sus grandes fortalezas, ya que permite concentrarse en la solución del problema en lugar de la sintaxis.

- Ampliable y empotrable: Como ya veremos más adelante en el presente documento, Python ofrece la posibilidad de crear extensiones que amplíen su funcionalidad en un determinado dispositivo, mediante la combinación con código C++. Así mismo, el código Python puede ser insertado e invocado dentro de un programa escrito en lenguaje C/C++.

- Interpretado: La ejecución de un programa Python es directa, puesto que no es preciso cargar o enlazar librerías, lo que deriva en un cierto grado de facilidad a la hora de portar nuestra aplicación a otro sistema que cuente con un intérprete Python apropiado. Cuando el código fuente de un programa Python es ejecutado por primera vez, se genera una forma intermedia denominada *bytecode* que mejora la velocidad de posteriores ejecuciones de la aplicación.

- Librerías: Esta es una de las características más interesantes de Python, y que lo dotan a su vez de una gran potencia para el desarrollo de aplicaciones, sus librerías estándar, las cuales ayudan al desarrollo rápido de aplicaciones complejas. En este sentido, Python posee extensiones para el manejo de: expresiones regulares, generación de documentos, evaluación de unidades, pruebas, procesos, bases de datos, navegadores web, CGI, FTP, correo electrónico, XML, XML-RPC, HTML, archivos WAV, criptografía, interfaz gráfica del usuario (GUI por sus siglas en inglés) usando Tk, así como otras funciones dependientes del sistema. De ahí deriva la típica expresión de que "Python viene con baterías incluidas" (ver figura 2.1).

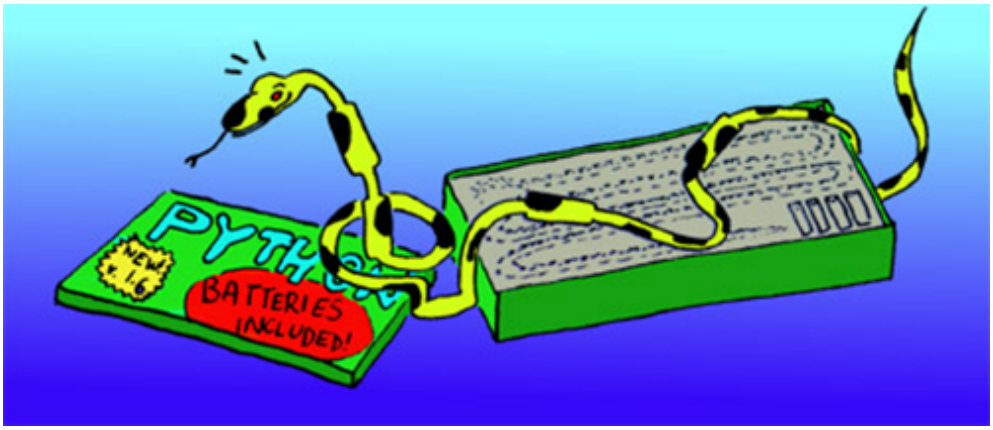

Figura 2.1: "Python viene con baterías incluidas".

A diferencia de los lenguajes compilados, Python se utiliza como lenguaje de programación interpretado, lo que plantea ciertas ventajas e inconvenientes.Por

un lado, los lenguajes compilados tienen la ventaja de que su ejecución suele resultar más rápida por lo general. Sin embargo, los lenguajes interpretados resultan más flexibles y portables.

Podríamos pensar que en un dispositivo con capacidad de cómputo limitada, como puede ser un teléfono móvil, esta característica podría acarrear problemas de lentitud al ejecutar las aplicaciones. Sin embargo, tenemos que tener en cuenta dos aspectos fundamentales capaces de mitigar en cierto modo el efecto anterior:

1. Cuando el intérprete de Python ejecuta una aplicación por primera vez, traduce el código fuente a una especie de pseudo-código máquina o *bytecode*, generando archivos `.pyc` o `.pyo`. Esto permite que las sucesivas ejecuciones de un programa se realicen a mayor velocidad.

2. En el caso particular de Python para terminales S60, se dispone de un conjunto de librerías estándar que facilitan la programación de aplicaciones para estos dispositivos, lo que implica que el código fuente desarrollado posea un número menor de instrucciones.

Por todas estas razonas, se ha decidido desarrollar el actual proyecto en lenguaje Python. Un lenguaje sencillo y potente, creado, mantenido y mejorado por una amplia comunidad de desarrolladores.

A partir de la versión 1.6.1 de este lenguaje, Python pasó a ser software libre, estando protegido por la licencia GNU GPL. Las sucesivas versiones están protegidas por la *Python Software Foundation License*, creada por la *Python Software Foundation*, una organización sin ánimo de lucro. Una característica que, como ya comentábamos anteriormente, representa otra ventaja de este lenguaje, ya que exime al desarrollador del pago de una posible licencia si desea utilizarlo para la creación de aplicaciones.

La versión Python 2.1 ya implementa reglas de ámbito (*scoping*), y la versión 2.2 unifica los tipos (escritos en C) y las clases (tipos escritos en Python) dentro de una misma jerarquía. Posteriormente, algunas decisiones sintácticas han sido influenciadas fuertemente por otros lenguajes de programación como Java. Actualmente las versiones de Python más utilizadas en los diferentes proyectos son la 2.5, 2.6 y la reciente 3.0. A día de hoy, Python se desarrolla como un proyecto de código abierto y es administrado por la *Python Software Foundation*, siendo la 3.0.1 la última versión estable.

Concretamente, en el presente estudio se utilizará Python 2.6 para la implementación de la aplicación del equipo receptor, así como PyS60 dotado con núcleo Python 2.x para la realización de la aplicación móvil.

Introducción a la sintaxis Python

Podemos resumir alguno de los conceptos más importantes sobre la sintaxis de Python [6] en la tabla 2.3.

Elemento	DescripciónTexto

Variables	En Python no es necesario declarar el tipo de datos que una variable va a almacenar. Python lo gestiona de forma dinámica. Los nombres de las variables pueden tener una longitud arbitraria, pudiendo contener letras y números. Los nombres de variables siempre deben empezar con una letra. Se considera que una letra minúscula es diferente de la misma letra mayúscula.
	
	`a=7`
	`a="Hola"`

Strings o cadenas de caracteres	`print "Hola %s, Hola %s" ("Mundo", Python)` El resultado mostrado por pantalla será: Hola Mundo, Hola Python
Comillas dobles y comillas simples	`"Python"` `'Python'` En Python las comillas dobles y las simples sirven para lo mismo. No existe diferencia entre el uso de una u otra.
Listas	`nombres = [u"Python", u"PyS60", u"Symbian"]` Son conjuntos ordenados con una serie de valores almacenados, encerrados entre corchetes.
Tuplas	`(1,2,3,4)` tupla de 4 elementos Una tupla es una lista inmutable (no se puede cambiar a lo largo de la ejecución del programa). Los elementos de la tupla se separan con comas y los valores se encierran entre paréntesis.
Diccionarios	`{'jamón':2, 'queso':7}` Son estructuras tipo `"clave:valor"`. Cada clave tiene asociado un valor determinado. Estas parejas se encierran entre llaves.
Comentarios	`# Esto es un comentario` Los comentarios son líneas no ejecutables del programa. El intérprete de Python ignora los comentarios, pero son de gran utilidad para documentar el programa. Empiezan con almohadilla y terminan en el final de la línea.

Condición if	`if x>0:`
	`  # Realiza la acción 1`
	`elseif x==2:`
	`  # Realiza la acción 2`
	`elseif x==3:`
	`  # Realiza la acción 3`
	`else:`
	`  # Realiza la acción 4`

Bucle for	`nombres = [u"Python",u"PyS60",u"Symbian"]`
	`for elemento in nombres:`
	`  print elemento`
	La sintaxis del bucle `for` debe leerse como: "para cada elemento en (la lista de) nombres imprime (el nombre del) elemento"

Cuadro 2.3: Conceptos básicos en Python

2.2.2. PyS60

Actualmente se tiene la creencia de que desarrollar una aplicación móvil es similar a desarrollar una aplicación tradicional "en miniatura". Pero la implementación de software para este tipo de dispositivos conlleva el uso de un conjunto concreto de herramientas específicas que consigan aprovechar las capacidades del terminal, así como la utilización de un lenguaje de programación adecuado.

Como ya comentamos anteriormente, el sistema operativo Symbian y la plataforma S60 se encuentran actualmente disponibles en una amplia gama de dispositivos en todo el mundo. Symbian OS se basa en el lenguaje de programación Symbian C++, basado a su vez en C++ pero con una serie de particularidades. Las aplicaciones nativas para este tipo de terminales se implementan en este lenguaje, dando como resultado un software eficiente y con pleno acceso a todos los recursos y funcionalidades del dispositivo.

En los últimos años, los teléfonos móviles están incluyendo una gran cantidad de nuevas características tales como: *bluetooth*, acelerómetro, GPS, acceso a la red, sensores, etc. Por tanto, se necesitan herramientas adecuadas para explotar

estas nuevas características, y que a su vez permitan una posible ampliación en el supuesto de que surjan nuevas funcionalidades para dichos dispositivos.

Por desgracia, debido a las particularidades de la programación en Symbian C++, la curva de aprendizaje de este lenguaje posee una elevada dificultad, ya que requiere del uso de técnicas especiales, como pueden ser el uso de descriptores o la limpieza de la pila. Esto puede hacer que, incluso aplicaciones relativamente sencillas, sean bastante complicadas de programar, sobre todo si comparamos su implementación con el desarrollo en otros entornos.

En este contexto, es posible que muchas de las técnicas desarrolladas en los años noventa para un conjunto restringido de hardware de terminales móviles, sólo provoque una complejidad innecesaria en el código fuente, obligando a los programadores a concentrarse en las rutinas de bajo nivel en lugar de las aplicaciones y sus características específicas. Desde este punto de vista, las técnicas de Symbian, como el paradigma de la gestión de memoria, reportan un dudoso beneficio al programador de aplicaciones móviles.

A priori podríamos pensar que en este entorno resulta difícil realizar un movimiento hacía un estilo de programación de más alto nivel, como marca el paradigma de programación moderno. No obstante, y a pesar de que Symbian C++ es un lenguaje muy especializado, los dispositivos dotados de Symbian OS permiten la creación de programas mediante multitud de lenguajes de alto nivel, como por ejemplo: Java ME, .NET, Python, Ruby, Flash Lite, Web Runtime (WRT) o el estándar C/C++.

Si bien lo ideal hubiera sido un estándar común de desarrollo para todos los dispositivos (como pudiera haber sido el lenguaje Java), la realidad nos dice que esto no es así, ya que los fabricantes han intentado realizar una optimización y perfeccionamiento de los distintos sistemas operativos y lenguajes sobre cada uno de sus dispositivos. Es por esto que, a día de hoy, nos encontramos con diversas plataformas para el desarrollo de aplicaciones móviles, lo que en ocasiones trae consigo ciertos inconvenientes de portabilidad, efectividad y mantenimiento de aplicaciones.

Desde este punto de vista, parece conveniente la utilización de algún lenguaje de programación capaz de explotar las capacidades de la plataforma tecnológica S60, y que a su vez permita el desarrollo rápido de aplicaciones, una característica importante si tenemos en cuenta la rápida evolución del mercado en el que nos movemos. Un lenguaje de programación que permita esconder los complejos detalles técnicos de bajo nivel tras una simple interfaz, ofreciendo la posibilidad de desarrollar aplicaciones totalmente funcionales para terminales S60, con tan sólo unas pocas líneas de código.

Esta situación representa una de las motivaciones que ha llevado a que el presente trabajo se haya desarrollado mediante el lenguaje de programación Python, y más concretamente, en la parte móvil, en PyS60 [7] [16], una versión modificada y adaptada de dicho lenguaje para su uso en terminales de la serie 60 que disponen de sistema operativo Symbian.

Como ya comentamos al comienzo de la sección 2.2, el presente proyecto ha sido desarrollado enteramente mediante el lenguaje de programación Python. Como ya veremos en capítulos posteriores, el sistema implementado consta de dos partes fundamentales. Por una parte, una aplicación móvil capaz de enviar un flujo M-JPEG desde un terminal. Por otro lado, un equipo receptor capaz de reproducir dicha secuencia de imágenes por pantalla. En este sentido, cabe destacar que mientras que la parte receptora ha sido desarrollada en su totalidad mediante el lenguaje de programación Python, la parte móvil utiliza PyS60.

PyS60 proporciona al desarrollador la posibilidad de acceder de forma sencilla a ciertos recursos hardware del dispositivo, además de ofrecer todas las ventajas de Python a nivel de lenguaje de programación.

En este trabajo se ha utilizado la versión 1.4.5 de PyS60, basada en un núcleo Python 2.x, la cual ofrece la posibilidad de utilizar multitud de módulos de la biblioteca estándar de Python, además de incluir algunos módulos específicos para la plataforma móvil S60, como por ejemplo:

- Módulos para el manejo de *widgets* o elementos nativos del GUI, así como otros elementos de la interfaz gráfica.
- Acceso a las capacidades *Bluetooth* del terminal.
- Establecimiento de conexiones de datos GPRS.
- Acceso a la información de geolocalización suministrada por el dispositivo GPS del terminal.
- Control de mensajería SMS.
- Información del sistema y características del terminal móvil (hardware, modelo, etc.).
- Acceso a la cámara de fotos del terminal.

Cabe destacar que PyS60 hace uso de las APIs C++ utilizadas en Symbian, y dispone de multitud de módulos, ya desarrollados, dentro de sus librerías estándar,

con diversas funcionalidades adicionales para dispositivos móviles específicos. Sumado a esto, y con el objetivo de extender las funcionalidades, la comunidad de desarrolladores puede crear extensiones de Python en C++ basándose en el API de Symbian.

Sin duda alguna, este es uno de los puntos fuertes de utilizar Python, el cual permite agregar nuevas funcionalidades al lenguaje mediante la simple acción de importar módulos, que no son más que mini-códigos capaces de llamar a los recursos del sistema.

Para poder disponer de PyS60 en nuestro terminal móvil, es preciso instalar en el dispositivo el intérprete de Python para la plataforma S60, el cual será el encargado de ejecutar el código fuente de la aplicación móvil que desarrollemos.

Como ya comentamos anteriormente, PyS60, al igual que Python, proveen soluciones tipo *script*, las cuales proporcionan gran flexibilidad y capacidades para desarrollar nuevos programas con rapidez, pero que por contra no generan aplicaciones binarias nativas.

Un aspecto negativo de esta característica, podría ser el hecho de que las soluciones "no nativas" siempre generan un código menos eficiente que el de las aplicaciones nativas.

En contraposición a este hecho, en los últimos años, la capacidad de procesamiento y la memoria de los dispositivos móviles han aumentado significativamente, lo que de algún modo contrarresta esta carencia, posibilitando que muchos de los teléfonos móviles actuales puedan ejecutar un lenguaje interpretado como Python sin que el rendimiento del dispositivo se vea seriamente afectado. Sumado a esto, la extensión de los programas desarrollados en Python y PyS60 suele ser notablemente inferior a los implementados en código nativo, por lo que la línea que separa ambas fronteras se estrecha todavía un poco más.

Por tanto, las aplicaciones creadas mediante este lenguaje deberían poder ser ejecutadas en el terminal móvil con una velocidad aceptable, ofreciendo desde un primer momento un rápido acceso a funcionalidades complejas del terminal.

Todas las razones aquí descritas motivan que el lenguaje de programación Python, y más específicamente PyS60, resulten buenos candidatos para la creación de aplicaciones para dispositivos móviles Symban S60.

Módulos PyS60

Una de las características interesantes de PyS60 es que, además de contar con la biblioteca estándar de Python, cuenta con módulos específicos que per-

miten interactuar con las capacidades de esta plataforma móvil [8]. Python trae consigo una gran cantidad de módulos estándar, cuya documentación puede ser consultada en la web oficial de Python [5], mientras que PyS60 incluye una serie de módulos propietarios, específicos para teléfonos móviles, los cuales solo están disponibles para la plataforma S60.

En PyS60, al igual que en Python, un módulo es una colección de funciones relacionadas que se agrupan en un mismo fichero. Todos los módulos incluidos en Python y PyS60 son instalados junto con el entorno de ejecución.

En el cuadro 2.4 podemos observar una lista con la mayoría de los módulos específicos para teléfonos móviles incluidos en PyS60. En ella se presenta de forma resumida las características y funcionalidades que cada módulo ofrece al desarrollador.

Módulo S60	Descripción
appuifw	Proporciona una interfaz para las aplicaciones S60: Cuadros de diálogo, listas de selección, notas, *pop-up*, etc.
audio	Permite la reproducción y grabación de ficheros de audio y provee de acceso a al motor *text-to-speech*.
messaging	Proporciona APIs para el envío de SMS y MMS.
inbox	Proporciona herramientas para la lectura de SMS entrantes en el terminal, así como su contenido, tiempo de llegada y número del emisor. También permite borrar SMS.
camera	Permite tomar fotografías y activar/apagar el visor de imágenes.
graphics	Proporciona acceso a las primitivas para gráficos 2D, así como la carga de imágenes en el terminal. Permite realizar acciones como el guardado de imágenes, redimensionado y transformación.
sysinfo	Ofrece un API para comprobar la información del sistema del terminal móvil S60: Nivel de batería, código IMEI, nivel de señal o espacio disponible en memoria.
gcanvas	Proporciona una interfaz de usuario para la presentación de gráficos OpenGL ES.
gles	Proporciona una capa entre Python y los gráficos OpenGL ES 3D/2D.

location	Ofrece un API de servicios de localización: Lectura de *Cell-ID*, etc.
positioning	Proporciona un acceso básico a la información de localización geográfica del terminal.
socket	Extensiones S60 para el módulo estándar de Python para *sockets*, protocolo Bluetooth RFCOMM y OBEX, configuración por defecto del punto de acceso.
urllib	Proporciona un interfaz de alto nivel para la carga de datos a través de la *World Wide Web*.
httplib	Proporciona una interfaz con el protocolo HTTP.
telephone	Proporciona un API para manejar diversas funcionalidades del teléfono como el marcado o el descolgado.
calendar	Proporciona un API para el manejo de los servicios de calendario: Lectura, creación de entradas, configuración de alarmas.
contacts	Proporciona un API para el manejo de los servicios del libro de direcciones, permitiendo la creación de una base de datos con información de nuestros contactos.
e32	Proporciona utilidades relacionadas con el sistema operativo Symbian que no están referidas al interfaz del usuario y que no son proporcionadas por la librería de módulos estándar de Python.
e32db	Proporciona un API para la manipulación de bases de datos relacionales con una sintaxis SQL limitada.
keycapture	Ofrece un API para la captura global de eventos de teclado.
topwindow	Interfaz para la creación de ventanas que vayan a ser mostradas en la parte superior de otras aplicaciones.

Cuadro 2.4: Módulos Python para terminales S60

En el desarrollo de este proyecto utilizaremos algunos de estos módulos para

implementar y poner en marcha nuestra aplicación móvil.

Seguridad en la plataforma Symbian y Python para S60

En Symbian existe un marco de trabajo que limita lo que las aplicaciones pueden hacer en un terminal basado en la versión 9 o superior de este sistema operativo. Para dispositivos S60, esta plataforma de seguridad ha sido incluida a partir de la 3ª Edición, por lo que, a la hora de desarrollar aplicaciones para este entorno, es necesario tener en cuenta una serie de consideraciones. Cabe destacar que el apartado de la seguridad excede los límites del presente documento, aunque sí puede resultar interesante ofrecer al lector una breve introducción sobre algunas cuestiones.

Existen diferentes métodos para ejecutar una aplicación móvil desarrollada mediante Python y PyS60. Por ejemplo, un desarrollador puede empaquetar una aplicación Python en un fichero auto-firmado `SIS`, el cual podrá ser instalado en cualquier terminal móvil que permita la instalación de aplicaciones que no sean de confianza o potencialmente no seguras, o si lo prefiere se podrá proporcionar la aplicación en un fichero plano o *script* `.py` para los usuarios avanzados, de tal forma que pueda ser ejecutado por el intérprete Python instalado en el dispositivo. En nuestro caso, este último método ha sido el elegido para ejecutar la aplicación móvil implementada a lo largo del presente trabajo.

Otra manera de acceder a las aplicaciones PyS60 será mediante el uso de un conjunto de capacidades garantizadas para el usuario, como pueden ser: `LocalServices`, `NetworkServices`, `UserEnviroment`, `ReadUserData` y `WriteUserData`. Para esta última forma se recomienda visitar la documentación PyS60 de referencia proporcionada en en el Wiki de Nokia [7].

Como desventaja, podemos citar algunos ejemplos de funcionalidades que no son accesibles utilizando el juego de capacidades incluido:

- Acceso a la información del identificador de celda (función `location.gsm_location()`). Requiere capacidades `ReadUserData`, `ReadDeviceData` y `Location`.

- Acceso a la captura global de eventos de teclado (módulo `keycapture`). Requiere capacidad `SwEvent`.

- Configuración de la hora del dispositivo (función `e32.set_home_time()`). Requiere capacidad `WriteDeviceData`.

Si se requiere del acceso a las capacidades contenidas en el conjunto garantizado para el usuario conviene visitar Symbian Signed [22] para obtener una información más detallada.

2.2.3. Base de datos SQLite

Como ya comentamos en la introducción de este documento (ver capítulo 1), el diseño implementado en este trabajo plantea un sistema formado por dos tipos de usuarios diferenciados. Los usuarios móviles, capaces de enviar flujos M-JPEG desde sus terminales, y los usuarios de los equipos receptores, encargados de reproducir dichas secuencias de imágenes. Además, uno de los objetivos del presente proyecto consiste en dotar a dicho sistema de soporte para la movilidad en Internet mediante el protocolo SIP, con lo que si en mitad de una sesión de datos multimedia entre un terminal móvil y un equipo receptor, se produce un cambio de red por parte del primero, el sistema sea capaz de actualizar la información de conexión referente al flujo de datos M-JPEG enviado por el dispositivo móvil, y su reproducción pueda continuar en el equipo receptor.

En este proceso de actualización de los parámetros de sesión, existe cierta información vital acerca de dicha transacción que debe ser guardada convenientemente por el equipo receptor para posteriores consultas. Concretamente, en este trabajo se ha decidido almacenar dichos datos en una base de datos SQLite [12].

Existen problemas para los que guardar nuestros datos en ficheros de texto plano, en archivos XML, o mediante serialización con los módulos Python `pickle` o `shelve` pueden resultar soluciones poco convenientes. En ocasiones no queda más remedio que recurrir a las bases de datos, ya sea por cuestiones de escalabilidad, de interoperabilidad, de coherencia, de seguridad, de confidencialidad, etc.

Si trabajamos con ficheros de texto, deberemos almacenar la información en formato plano, lo que implicará el diseño de una estructura para guardar los diferentes campos de información requeridos por la aplicación, así como establecer un orden determinado, crear un diccionario, etc.

Por otro lado, una simple tabla en memoria es una solución rápida, pero volátil, además de seguir implicando el diseño de una estructura propietaria.

Si trabajamos con una base de datos, la estructura interna estará prefijada, pudiéndose crear tablas con los campos de información que se necesiten, posibilitando una futura ampliación mucho más sencilla.

Aunque en el presente proyecto, las pruebas ejecutadas se han realizado me-

diante un solo terminal y un único usuario, se ha decidido trabajar con bases de datos en Python por varias razones:

1. La base de datos SQLite ya está incluida en la propia instalación de Python para PC a partir de la versión 2.5 del mismo, por lo que, con esta decisión aprovecharíamos parte del potencial que Python ofrece al desarrollador.

2. SQlite proporciona una infraestructura madura, ligera, que consume muy pocos recursos, por lo que el rendimiento general del sistema no se ve afectado.

3. Proporciona un acceso rápido a la información almacenada mediante el estándar de consultas SQL.

4. Aunque el trabajo con ficheros de texto plano o XML es sencillo, la utilización de bases de datos en Python tan solo requiere un manejo elemental de SQL, y nos evita la creación de una solución propietaria para el almacenamiento y consulta de la información del sistema.

5. No necesita instalar y ejecutar un proceso servidor independiente que facilite la comunicación con el programa implementado, ya que se trata de una pequeña librería en C que se integra con la aplicación.

6. La misma versión Python también incorpora un módulo denominado `sqlite3` [12] compatible con esta base de datos, por lo que no necesitaremos ningún tipo de configuración extra.

7. No volatilidad de la información, lo que permitiría en un diseño más avanzado sobreponerse al reinicio o caída del equipo.

8. El sistema implementado resulta significativamente más escalable ante un aumento de usuarios del servicio.

9. El sistema implementado puede actualizarse con mayor facilidad, permitiendo la inclusión de nuevos campos de información para ser almacenados como por ejemplo: formato de las imágenes enviadas, *codecs* utilizados, información sobre los usuarios del sistema, etc.

10. La gestión de los usuarios resulta mucho más sencilla y funcional, al evitar el manejo intensivo de *scripts* que procesen las cadenas de texto de dichos ficheros.

Existen cientos de bases de datos en el mercado, tanto comerciales como gratuitas. También existen decenas de módulos distintos para trabajar con dichas bases de datos en Python, lo que significa que el desarrollador tiene a su disposición decenas de APIs distintas para aprender.

En Python, como en otros lenguajes de programación como Java con JDBC, existe una propuesta de API estándar para el manejo de bases de datos, de forma que el código sea prácticamente igual, independientemente de la base de datos que estemos utilizando por debajo. Esta especificación recibe el nombre de Python *Database* API o DB-API y se recoge en el PEP 249 [13]. DB-API se encuentra en estos momentos en su versión 2.0, y existen implementaciones para las bases de datos relacionales más conocidas, así como para algunas bases de datos no relacionales.

Uso básico de DB-API

Antes de comenzar a trabajar con `sqlite3` y mostrar su funcionamiento, conviene destacar algunas características interesantes sobre este módulo Python [14].

Todos los controladores o *drivers* compatibles con DB-API 2.0 deben tener tres variables globales que los describen. A saber:

- `apilevel`, una cadena con la versión de DB API que utiliza. Actualmente solo puede tomar el valor "1.0" o "2.0". Si la variable no existe se asume que es "1.0".
- `threadsafety`, un simple entero que toma valores de 0 a 3 y sirve para describir el nivel de seguridad que ofrece el módulo para su uso en programación con hilos o *threads*.
    - Si toma el valor 0, no se puede compartir el modulo entre *threads* sin utilizar algún tipo de mecanismo de sincronizacion que asegure un acceso y compartición segura de los datos.
    - Si toma el valor 1, pueden compartir el modulo pero no las conexiones.
    - Si toma el valor 2, módulos y conexiones pero no cursores.
    - Y por último, si toma el valor 3, es totalmente *thread-safe*.
- `paramstyle`, informa sobre la sintaxis a utilizar para insertar valores en la consulta SQL de forma dinámica.
    - `qmark` o interrogaciones.

```
sql = ‘‘select all from t where valor=?’’
```

- `numeric`, un número indicando la posición.

```
sql = ‘‘select all from t where valor=:1’’
```

- `named`, el nombre del valor.

```
sql = ‘‘select all from t where valor=:valor’’
```

- `format`, que contiene especificadores de formato similares a los de la función `printf` del lenguaje C.

```
sql = ‘‘select all from t where valor=%s’’
```

- `pyformat`, similar al anterior, pero con las extensiones de Python.

```
sql = ‘‘select all from t where valor=%(valor)’’
```

Concretamente, los valores correspondientes a la versión de `sqlite3` utilizada en este proyecto son los siguientes:

```
>>> import sqlite3 as dbapi
>>> print dbapi.apilevel
2.0
>>> print dbapi.threadsafety
1
>>> print dbapi.paramstyle
qmark
```

Lo que implica que, la versión de `sqlite3` utilizada en el sistema implementado, requiere un nivel de seguridad con dos reglas a seguir:

1. No compartir las conexiones.

2. Realizar las consultas de forma dinámica a modo de interrogaciones.

Estas pautas han marcado la metodología seguida en este proyecto a la hora de trabajar con esta base de datos.

2.2.4. Librerías gráficas

El sistema implementado durante este trabajo, ha utilizado tres librerías para la creación de las interfaces gráficas de usuario desarrolladas para la aplicación del terminal móvil y la aplicación del equipo receptor:

- `appuifw` [8]: Módulo PyS60 utilizado exclusivamente en la aplicación móvil para crear la interfaz gráfica de usuario, así como los menús correspondientes. Este módulo incluido por defecto en la instalación de PyS60 para terminales móviles Symbian de la serie 60.

- `TkInter` [9]: Basada en la biblioteca gráfica `Tcl/Tk`, esta librería proporciona, en la aplicación implementada en el equipo receptor, el soporte necesario para la creación de interfaces multiplataforma, incluyendo entornos Windows, Macintosh, así como sistemas Unix. Esta librería está incluida por defecto en la instalación del intérprete Python estándar.

- `PIL` [10]: (*Python Imaging Library*), una librería externa para el lenguaje de programación Python que añade soporte para la apertura, manipulación y guardado de imágenes en diferentes formatos. Las funciones de dicha librería nos proporcionarán las herramientas necesarias para reproducir el flujo M-JPEG, procedente del terminal móvil o MS, en la aplicación implementada en el equipo receptor. Para más información sobre la instalación de esta librería adicional, se recomienda consultar el apéndice B.1.2 del presente documento.

2.3. Protocolos

2.3.1. SIP

En los últimos años, las recomendaciones de los diferentes foros técnicos sobre tecnología inalámbrica, han llevado a que el protocolo SIP (*Session Initiation Protocol*) [24] [25] esté presente en una cantidad significativa de dispositivos móviles, los cuales se han dotado recientemente con una arquitectura de protocolos de Internet. Un crecimiento de tendencia positiva, que se está viendo reforzado con la aparición en escena de los servicios de voz, basados en conmutación de paquetes o VoIP.

SIP es un protocolo de señalización desarrollado por el IETF *Working Group* con la intención de convertirse en el estándar para la iniciación, modificación y

finalización de sesiones interactivas de usuario, donde intervienen elementos multimedia como el vídeo, voz, mensajería instantánea, juegos en línea o la realidad virtual.

Concretamente, en este trabajo se ha implementado parcialmente la RFC 3261 [24] y la RFC 3311 [25] de este protocolo sobre UDP, con el objetivo de dotar al terminal móvil y al equipo receptor de las herramientas necesarias para establecer, actualizar y finalizar sesiones multimedia para el intercambio de información. En este sentido, uno de los objetivos principales de este proyecto consiste en implementar un sistema que permita la movilidad de sus usuarios mediante el protocolo SIP. Para ello, este trabajo se basa en el *Internet Draft* "Soporte para la movilidad multimedia mediante SIP" [23] .

SIP es un protocolo de señalización extremo a extremo que implica que toda la lógica debe ser almacenada en los dispositivos finales (salvo el rutado de los mensajes SIP), al igual que el estado de la conexión. El precio a pagar por esta capacidad de distribución y su gran escalabilidad es una sobrecarga en la cabecera de los mensajes, producto de tener que mandar toda la información entre los dispositivos finales.

De manera general, podemos resumir las características principales de SIP en la siguiente lista:

- Permite establecer, modificar y terminar sesiones multimedia entre dos o más participantes, e invitar a nuevos participantes a dichas sesiones.
- Funciona a nivel de aplicación dentro del modelo TCP/IP.
- Es independiente del protocolo de transporte sobre el que se transmite (TCP, UDP u otros).
- Utiliza mensajes de texto, siguiendo un modelo de "petición y respuesta" similar al de HTTP o al de SMTP.
- Utiliza un conjunto reducido de entidades que interactúan (*user agents* o agentes de usuario), así como varios tipos de servidores.
- El encaminamiento de los mensajes de SIP es completamente independiente del de los datos de las sesiones que establece.

Principalmente, SIP lleva a cabo cinco funciones básicas:

1. Determinar la localización de los puntos finales de la comunicación, es decir, los usuarios.

2. Contactar con dichos usuarios para determinar su disponibilidad y permitir el establecimiento de sesiones multimedia.

3. Establecer dichas sesiones multimedia mediante un intercambio de mensajes.

4. Modificar dichas sesiones de comunicación multimedia, así como la modificación de los parámetros de configuración de dichas sesiones, de tal forma que el proceso sea transparente para el usuario.

5. Finalizar sesiones multimedia.

Concretamente, el sistema implementado durante el presente proyecto será capaz de realizar las cuatro últimas funciones básicas definidas por el protocolo.

Cuando se establece una nueva sesión multimedia, SIP se encarga de gestionar el plano de señalización solamente, es decir, no transporta información multimedia, por lo que puede verse como un simple complemento de la comunicación. El inicio de la sesión, cambio o término de la misma, son independientes del tipo de medio o aplicación que se estará usando en la llamada; y una sesión puede incluir varios tipos de datos, incluyendo audio, vídeo y muchos otros formatos.

En el presente trabajo, el protocolo SIP se utilizará para establecer, actualizar y finalizar sesiones que involucren el envío y reproducción en destino de flujos M-JPEG capturados por la cámara por el terminal, o almacenados previamente en la memoria de un simulador software destinado a tal efecto.

Por otro lado, cabe destacar que SIP suele utilizarse en conjuntamente con otros protocolos:

1. Para la descripción del flujo de datos enviado desde el terminal móvil debe utilizarse otro protocolo denominado SDP (*Session Description Protocol*) (ver sección 2.3.3), cuyos campos se adjuntarán en los mensajes SIP intercambiados por el sistema.

2. SIP no proporciona la reserva de recursos para sesiones multimedia. Para ello se pueden usar otros protocolos como RSVP (*Reservation Protocol*) y extensiones sobre SIP. Aunque esta cuestión se sale de los límites de estudio del presente documento.

3. SIP no participa de ningún modo en la transmisión de la información multimedia. Por esta razón, el flujo de imágenes enviado desde el terminal móvil al equipo receptor se trasportará mediante el protocolo RTP (*Real-time Transport Protocol*) (ver sección 2.3.4) y el formato M-JPEG (*Motion Joint Photographic Experts Group*) (ver sección 2.3.5).

En el contexto de este trabajo, podemos considerar al protocolo SIP como uno de los pilares del sistema, ya que será el encargado de establecer las sesiones multimedia entre los usuarios móviles y los equipos receptores fijos para el envío y reproducción de flujos M-JPEG. Así mismo, SIP será el encargado de actualizar los parámetros de una sesión en marcha, en el supuesto de que el usuario móvil cambie de ubicación y de dirección IP, así como de finalizar dicha sesión cuando el usuario del terminal lo considere oportuno. Algunas de las premisas, que implica el hecho de utilizar el protocolo SIP para dar soporte de movilidad a una aplicación, son las siguientes:

- La aplicación debe permitir a los usuarios el establecimiento de sesiones multimedia extremo a extremo, independientemente del protocolo de red utilizado. Este procedimiento debe resultar transparente al usuario.
- En servicios de tiempo real señalizados mediante mensajes SIP, deberían optimizarse las rutas de los paquetes para mejorar el rendimiento de la aplicación. Aunque esta cuestión queda fuera de los límites del presente documento.
- Y lo más importante de todo, si el usuario se desplaza a una nueva red, el flujo de datos multimedia establecido previamente entre dos extremos debe poder ser reconfigurado y mantenido.

Sin embargo, también podemos citar algunas desventajas derivadas de este enfoque, como por ejemplo:

1. El uso de múltiples protocolos en un terminal o servicio puede incrementar en exceso la complejidad de su implementación.
2. Existen diferentes protocolos enfocados a la movilidad que realizan acciones similares mediante procedimientos diferentes, por lo que si decidimos implementar una solución deberemos prestar atención a la hora de compatibilizar las distintas redes. Tal es el caso del protocolo *Mobile IP* (el cual registra a los usuarios en un HA (*Home Agent*) o un FA (*Foreign Agent*)) y el protocolo SIP (el cual registra a los usuarios mediante un mensaje `REGISTER` y un servidor *proxy* de tipo *Registrar*).

A lo largo de este trabajo desarrollaremos diferentes procedimientos, mediante los cuales, el protocolo SIP será capaz de dotar a un terminal móvil de capacidades para la movilidad en Internet, así como los mecanismos necesarios para soportar esta característica. Así mismo, tanto la aplicación desarrollada para el terminal móvil, como la aplicación implementada en el equipo receptor, incluirán la lógica

de agente de usuario SIP necesaria para realizar las acciones de establecimiento, actualización y finalización de una sesión multimedia.

A continuación se enumeran las diferentes entidades SIP involucradas en el sistema desarrollado durante este trabajo, detallando si su implementación ha sido o no necesaria y la razón de dicha decisión.

Usuarios SIP

Un usuario SIP puede ser una aplicación de mensajería, un teléfono IP, y en general cualquier dispositivo o software que sea compatible con SIP y que tenga la capacidad de registrarse con una cuenta SIP. Los usuarios SIP reciben una URI (*Uniform Resource Identifier*) formada por "usuario"@"dominio", donde el campo dominio se corresponde con el servidor SIP donde se encuentra registrado.

Algunos ejemplos de direcciones SIP pueden ser:

- `sip:valvarez@uc3m.es`
- `sip:valvarez@163.117.141.227:5060`

Concretamente, y puesto que el presente trabajo se centra en los extremos del sistema, se obviará el procedimiento de registro SIP (`REGISTER`) que involucra a una tercera entidad conocida como *proxy* SIP, aunque se dotará a ambos extremos (teminal móvil y equipo receptor) de un usuario SIP ficticio que permita realizar una implementación coherente con lo establecido en la RFC 3261 [24].

Agentes de usuario SIP

En cada iteración del intercambio de mensajes SIP en una sesión multimedia participan los denominados *User Agents* o "Agentes de usuario SIP". Cada usuario SIP utiliza un *User Agent* para enviar y recibir mensajes SIP.

Son muchos los dispositivos que pueden proporcionar un *User Agent*: teléfonos móviles, teléfonos fijos, computadores, televisores, etc. Existiendo dos tipos:

1. UAC (*User Agent Client*), aplicación cliente encargada de realizar peticiones SIP, denominadas *request*.
2. UAS (*User Agent Server*), aplicación servidora que tiene la misión de interactuar con el usuario cuando se recibe una petición SIP y contestar dichas peticiones generando las respuestas correspondientes.

Concretamente, en el presente trabajo se implementará un agente de usuario SIP en cada extremo del sistema (uno en la aplicación móvil y otro en la aplicación del equipo receptor). Estos agentes de usuario contarán con la lógica necesaria para establecer, actualizar y finalizar sesiones multimedia entre ambos extremos de la aplicación.

Servidores SIP

Un servidor SIP es una aplicación o dispositivo que permite crear y gestionar cuentas SIP y permitir que los Usuarios SIP se registren almacenando la dirección IP donde deben acceder para realizar la comunicación con este usuario.

Existen distintos tipos de servidores SIP:

1. *Registrar Server*: Gestionan la información de localización de los usuarios en un determinado dominio.
2. *Redirect Server*: Este tipo de servidores pueden requerir autenticación por parte de los clientes que generan la petición. Se encarga de recibir solicitudes de clientes o procedentes de otros servidores SIP, obtener la asociación entre la URI SIP del usuario al que se intenta contactar y sus direcciones de contacto actualizadas, y devuelve dichas direcciones de contacto al usuario que realizó la petición originaria.
3. *Proxies*: Actúan encaminando solicitudes hacia UAS, o encaminando respuestas hacia otros UAC. De hecho, en la práctica suele haber varios *proxies* en el camino de una determinada solicitud, hasta llegar del UAC origen al UAS destino. Las respuestas SIP siguen el mismo camino de vuelta, es decir, atravesando los mismos *proxies*. Los *proxies* pueden tomar decisiones más avanzadas como: elegir un nuevo encaminamiento para una determinada solicitud, modificar una determina solicitud, validar, redirigir, duplicar, autenticar usuarios, eliminar mensajes, etc.

El presente trabajo, y como ya se ha mencionado anteriormente, se centra en el estudio de los extremos de la comunicación, suponiéndose siempre una situación de registro previo en un servidor SIP predeterminado para dichos extremos del sistema.

2.3.2. Movilidad SIP

El presente apartado, y una parte fundamental de este trabajo, está basado en el borrador de IETF "Soporte para la movilidad multimedia con SIP" [23], ya que

uno de los objetivos principales del presente proyecto, consiste en desarrollar un sistema que permita la movilidad en Internet de los usuarios mediante el protocolo SIP.

Inicialmente, el diseño propuesto consta de dos tipos de equipos o estaciones, un terminal móvil S60 capaz de enviar un flujo M-JPEG sobre RTP capturado a través de su cámara integrada (o almacenado previamente en memoria en el caso de utilizar un emulador software S60), y una estación receptora capaz de reproducir y visualizar por pantalla la información multimedia recibida.

Como ya comentamos en la sección 2.3.1, el protocolo SIP es capaz de proveer a un terminal móvil de capacidades *hand-off*, o dicho de otro modo, proporciona soporte para el cambio de red del terminal, mientras que la comunicación continúa de forma transparente para el usuario.

A lo largo de las próximas secciones detallaremos la arquitectura móvil, así como los elementos radio y de red implicados en un caso práctico real, con el objetivo de obtener una visión general de nuestro marco de trabajo. Por otra parte, se detallará brevemente el procedimiento SIP para realizar un *hand-off* o cambio de red, así como el intercambio de mensajes SIP realizado entre el terminal móvil y el equipo receptor, para la correcta actualización de los parámetros de una sesión multimedia en curso.

Arquitectura para la movilidad de terminales multimedia basada en el protocolo SIP

El desarrollo de aplicaciones multimedia que permitan la movilidad en Internet, se sustenta mediante la infraestructura GPRS y las redes IP tradicionales. Por ende, antes de afrontar otras cuestiones técnicas acerca de este trabajo, resulta recomendable revisar brevemente los elementos involucrados en el soporte de esta característica.

El presente epígrafe es meramente aclaratorio, pero nos servirá para introducirnos en el ámbito de las aplicaciones multimedia con soporte para la movilidad en Internet, el marco donde se encuadra la aplicación que desarrollaremos mediante lenguaje Python, así como los módulos PyS60 disponibles para la plataforma S60, cuyo funcionamiento y características detallaremos en sucesivos capítulos.

La figura 2.2 muestra, de forma esquematizada, la arquitectura general de una plataforma extremo a extremo con soporte para la movilidad en Internet de terminales multimedia. Este tipo de estructuras de red poseen dos funciones fundamentales:

1. Proporcionar acceso inalámbrico a la red, basado en el protocolo IP, a todos los usuarios autorizados.

2. Proporcionar una infraestructura cableada, o *backbone*, basada en conmutación de paquetes extremo a extremo para el transporte de datos.

Por otro lado, y como podemos observar en la figura 2.2, en la parte de las estaciones móviles se proporciona un acceso de red inalámbrico mediante dos elementos: RAN (*Radio Access Network*) y ERC(*Edge Router & Controller*).

Este acceso inalámbrico es el encargado de interactuar con las entidades de control de la red, denominadas DCA (*Domain Control Agent*) (ver figura 2.2).

Con el objetivo de clarificar todos estos conceptos, a continuación vamos a describir brevemente los elementos o entidades de la figura 2.2, así como sus funcionalidades. Dado que los detalles de la arquitectura GSM/GPRS no son objeto del presente proyecto, no se profundizará más en ellos en el resto del documento, aunque resulta recomendable su definición para clarificar posteriores menciones a los mismos.

MS (*Mobile Station*)

Es el terminal móvil, el cual permite a un cliente autorizado comunicarse con el resto de usuarios de la red, así como establecer sesiones de intercambio de datos multimedia o llamadas de voz.

Centraremos nuestra atención en este elemento más adelante.

RAN (*Radio Access Network*)

Aunque no es objeto del presente estudio, conviene aclarar que la RAN permite a un MS el acceso radio a la red inalámbrica, así como a las infraestructuras de la red cableada fija.

Generalmente, la RAN está formada por un conjunto de estaciones base (BS), así como una estación base de control (BSC). Así mismo, las RANs suelen estar programadas con un software radio, siendo deseable que su implementación sea independiente de la tecnología hardware subyacente, con el objetivo de minimizar los requisitos o restricciones para su despliegue.

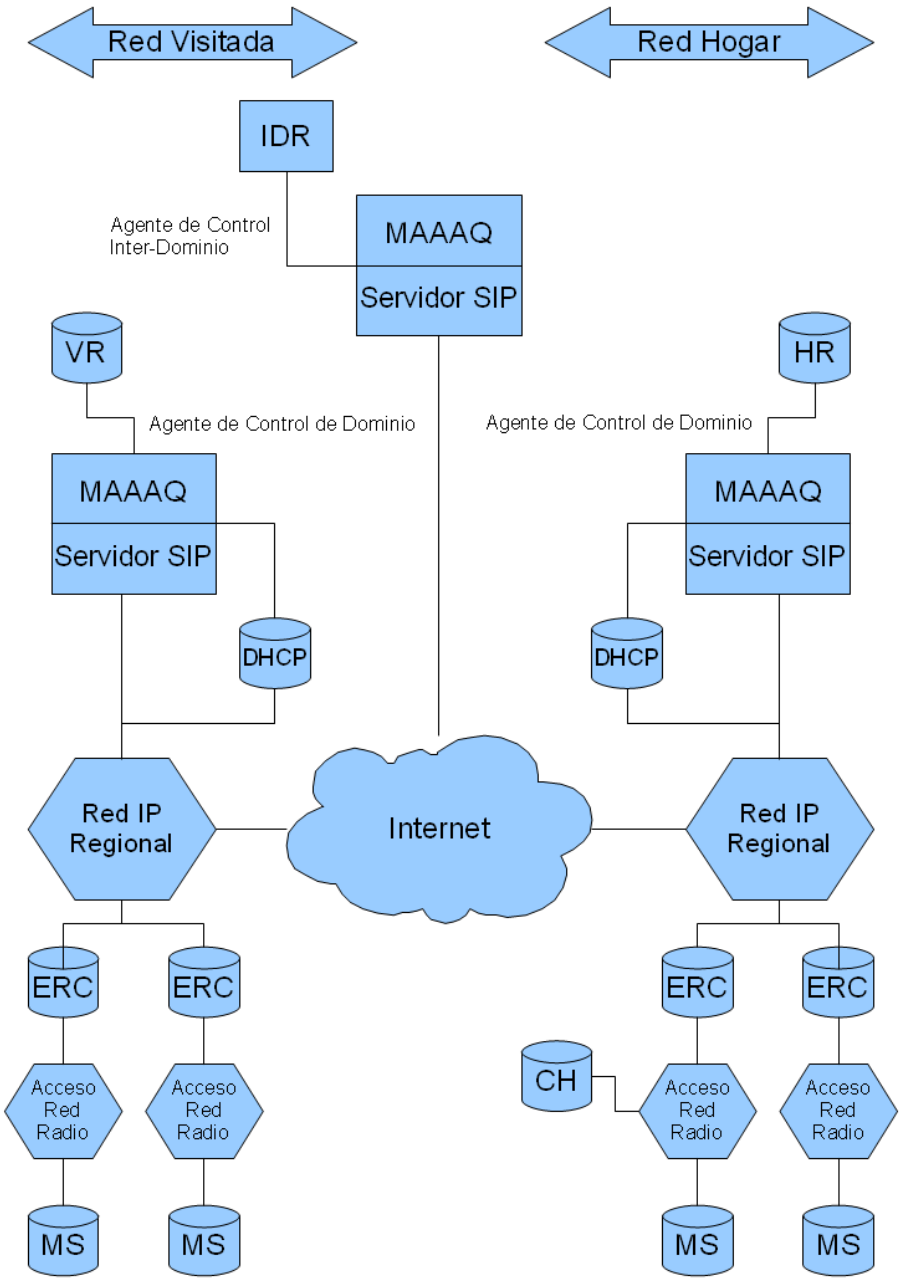

Figura 2.2: Arquitectura para movilidad en Internet.

ERC (*Edge Router & Controler*)

El ERC es un sistema de control y enrutamiento que conecta un acceso inalámbrico con una red IP cableada regional.

Un ERC está compuesto por dos entidades funcionales, un ER (*Edge Router*) encargado de del enrutamiento de paquetes IP, y un ECA (*Edge Control Agent*) el cual implementa un agente de control inteligente que se encarga de interactuar con un DCA (*Domain Control Agent*) con el objetivo de controlar los RANs, así como diversas tareas de control en redes de cierta magnitud.

Aunque en la figura 2.2 se observa que cada RAN va conectado a un ERC, en la práctica un ERC se encarga de controlar varios RANs.

DCA (*Domain Control Agent*)

El DCA proporciona un gestor de sesión que permita la interacción entre los usuarios y los sistemas de control de la red. Este elemento es el encargado de proporcionar las capacidades MAAAQ:

- Gestión de movilidad.
- Autenticación, autorización y gestión de cuentas. Estas características se conocen con el acrónimo AAA (*Authentication, Authorization & Accounting*).
- Gestión de QoS o calidad de servicio.

Como veíamos en la figura 2.2, las entidades DCA hogar y el DCA visitado interactuan directamente, o mediante un tercero o IDCA (*Inter-Domain Control Agent*), que asume el papel de intermediario de confianza entre los dos anteriores.

Procedimiento *hand-off* o cambio de red

En un terminal real, cuando un usuario cambia de ubicación y el sistema detecta que el nivel de señal recibido es insuficiente, la aplicación realiza un cambio de subred intradominio o de red interdominio, con lo que el terminal recibe una nueva dirección IP a partir de un servidor DHCP determinado. Este cambio implica que el terminal móvil tenga que notificar al equipo receptor la nueva ubicación lógica dentro de la red y actuar en consecuencia, produciéndose un proceso de intercambio de mensajes SIP entre el MS y el CH. Este procedimiento es conocido simplemente como *hand-off* o cambio de red.

Existen dos posibles tipos de cambio de red, y el esquema real de detección y reconocimiento de un *hand-off* se implementa desde el supuesto inicial de que un cambio de subred o de dominio implica un cambio de celda como pre-requisito:

1. *Hand-off* de subred: Se realiza cuando la estación móvil se desplaza de una subred a otra pero ambas pertenecientes al mismo dominio administrativo. La estación móvil interactúa con el servidor DHCP predeterminado para reconfigurarse automáticamente. Este proceso tiene una duración de un *round-trip* o viaje de ida y vuelta, es decir, el retardo de propagación entre los elementos implicados MS-DHCP-MS. La estación móvil vuelve a invitar al equipo correspondiente con su nueva dirección temporal, procedente de la asignación anterior. En estos casos, el mensaje `INVITE` del protocolo SIP debería incluir un campo `Record-Route` que contenga un identificador del servidor *proxy* de la red visitada.

2. *Hand-off* de dominio: Se realiza cuando la estación móvil se desplaza a una nueva red perteneciente a otro dominio administrativo. Exceptuando el hecho de que una estación móvil requiere un completo registro para completar el cambio de una red a otra, el procedimiento de *hand-off* entre distintos dominios administrativos es similar al procedimiento *hand-off* entre subredes descrito en el punto anterior. En estos casos, este procedimiento también es conocido como *roaming*.

De forma general, el intercambio de mensajes SIP que involucra el procedimiento de *hand-off* se ilustra en la figura 2.3.

Este procedimiento 2.3 será implementado como una de las funcionalidades principales del sistema práctico desarrollado durante este trabajo, y será explicado con mayor detalle en sucesivos capítulos del presente documento.

2.3.3. SDP

Como ya adelantábamos en la sección 2.3.1, el protocolo SIP no permite la descripción de sesiones multimedia, es decir, no permite describir las direcciones IP y los puertos a utilizar, qué codificadores usar, etc. Por esta razón, para la descripción del flujo de datos enviado desde el terminal móvil debe utilizarse otro protocolo denominado SDP (*Session Description Protocol*). Por esta razón, el presente proyecto implementa parcialmente parte de los mensajes SDP descritos en la RFC 4566 [26].

Concretamente, en este trabajo SDP es el encargado de enviar los parámetros de inicialización utilizando el rango de puertos UDP por encima del puerto 1024.

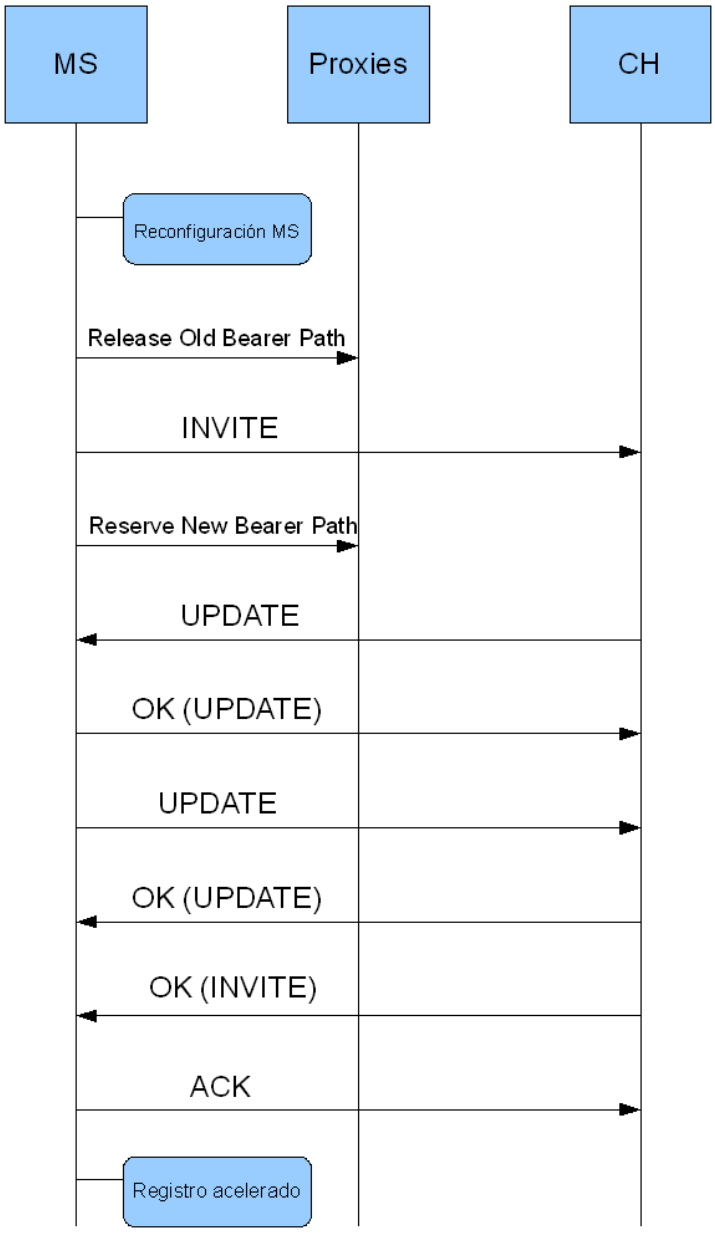

Figura 2.3: Señalización en un caso real para *hand-off* entre redes.

Los mensajes SDP serán enviados desde el terminal móvil al equipo receptor con el objetivo de que este último sepa la dirección y el puerto donde el flujo M-JPEG será enviado para poder reproducirlo correctamente por pantalla.

2.3.4. RTP

El protocolo RTP (*Real-time Transport Protocol*) es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio y vídeo en una vídeo-conferencia.

En el caso particular del sistema implementado durante el presente proyecto, el protocolo RTP será el encargado de trasportar los fragmentos de información procedentes de un flujo M-JPEG enviado desde el terminal móvil a la aplicación del equipo receptor. Para ello, se ha sido necesario implementar la RFC 3551 [27] sobre el protocolo UDP.

2.3.5. M-JPEG

Motion JPEG designa aquellos formatos multimedia donde cada fotograma o campo entrelazado de una secuencia de vídeo digital es comprimida por separado como una imagen JPEG (*Joint Photographic Experts Group*).

M-JPEG no es estrictamente un protocolo, sino un formato que permite transportar fragmentos de imágenes de una secuencia completa.

En el caso particular del sistema implementado, el terminal móvil envía una secuencia de imágenes capturadas directamente desde su cámara integrada, o previamente almacenadas en la memoria de un emulador software S60, al equipo receptor para su reproducción.

Los fragmentos de cada una de las imágenes de la secuencia enviada se transportan sobre el protocolo RTP (ver sección 2.3.4). Para ello, durante este trabajo ha sido necesario implementar la RFC 2435 [28].

El formato M-JPEG tiene una tasa de bits relativamente alta dada la calidad entregada, requiriendo más espacio de almacenamiento que otros formatos modernos para una calidad de imagen proporcionada. Desde este punto de vista, M-JPEG resulta una codificación ineficiente y desactualizada, pero suficiente para el caso de estudio que nos ocupa, donde el objetivo es poder señalizar un flujo de datos multimedia sencillo que permita implementar un sistema con soporte para la movilidad de usuarios basado en el protocolo SIP.

Capítulo 3

Descripción general del sistema

Este capítulo pretende proporcionar una visión general acerca del funcionamiento del sistema y el software desarrollado en el presente proyecto. Para ello, en los siguientes epígrafes se realizará una descripción de la arquitectura y de las funcionalidades proporcionadas por la aplicación implementada, así como una serie de sugerencias para futuros casos de uso.

Por otro lado, se presentan los requisitos funcionales del sistema, y se realiza una introducción a ciertos detalles de las herramientas seleccionadas para proporcionar el soporte requerido.

Al finalizar este apartado, debería haberse obtenido una noción general sobre la arquitectura y la funcionalidad del sistema propuesto, que permita abordar los capítulos posteriores del presente proyecto, en donde se profundizará en los detalles técnicos y de funcionamiento de la aplicación implementada, así como detalles acerca de la librería estándar de Python y los módulos PyS60 disponibles para la creación de aplicaciones multimedia para terminales S60.

3.1. Funcionalidad

3.1.1. Funcionamiento del sistema

El funcionamiento del sistema planteado puede resumirse de forma simplificada mediante la secuencia de ilustraciones 3.1, 3.2, 3.3, 3.4 y 3.5.

El objetivo de la aplicación MS consiste en enviar un flujo M-JPEG capturado directamente desde la cámara integrada en el terminal móvil. El equipo CH, destinatario de dicha secuencia de imágenes, se encargará de reproducirlo por

pantalla para que el usuario receptor pueda visualizarlo.

Como se puede observar en la figura 3.1, inicialmente el terminal móvil (MS o *Mobile Station*), que se encuentra ubicado en una red o dominio determinado, y recibe cobertura de señal a través de una BTS (*Base Transceiver Station*), decide establecer una sesión para el intercambio de datos multimedia con un equipo IP fijo (CH o *Corresponding Host*) ubicado en otra red distinta, utilizando para ello los diferentes elementos de la red inalámbrica móvil y red fija.

Aunque por simplicidad esquemática en la figura 3.1 no aparece, obsérvese que al igual que el equipo CH, el MS tiene integrado un agente de usuario SIP, que le permitirá realizar un intercambio de mensajes con CH para el establecimiento de una sesión multimedia.

Por otro lado, cabe destacar que, tanto MS como CH, ya se encuentran registrados mediante el procedimiento SIP `REGISTER` en su servidor *proxy* predeterminado, pero aunque los módulos Python implementados en este trabajo permiten dicho mecanismo, este procedimiento queda fuera de los límites del presente estudio, ya que el software desarrollado en este proyecto se centra en los extremos finales del sistema (MS y CH) y no en las entidades SIP intermediarias. Por esta razón, y aunque se presupone su existencia, en la figura 3.1 no aparece ninguna entidad del tipo *proxy* SIP.

Llegados a este punto, el terminal móvil decide iniciar una sesión multimedia con CH. Para ello, el MS utiliza el protocolo SIP y envía un mensaje `INVITE` al equipo receptor, el cual permite establecer dicha sesión multimedia e iniciar el intercambio de información entre los dos equipos.

Por su parte, CH comprueba y registra el identificador de usuario, enviado por el MS, así como la información de conexión, el tipo de codificación de la información enviada y otros datos de interés en una base de datos. Si este proceso resulta satisfactorio, el CH registra la conexión donde el terminal móvil desea enviar su flujo multimedia y comienza su reproducción en pantalla para que el usuario receptor pueda visualizarlo.

Como ya se justificó en la sección 2.2.3 del presente documento, por cuestiones de escalabilidad, así como otras ventajas, la gestión de sesiones y flujos M-JPEG activos mediante una base de datos ligera, parece algo recomendable. De ahí, el motivo de que CH disponga de la lógica necesaria para interactuar con una base de datos SQLite, con el objetivo de controlar que usuarios y conexiones están activas, a la par que poder diferenciar entre distintos flujos de imágenes procedentes de diferentes terminales, decidiendo qué imágenes reproducir en el equipo.

Como se puede observar en la figura 3.2, una vez establecida la sesión SIP

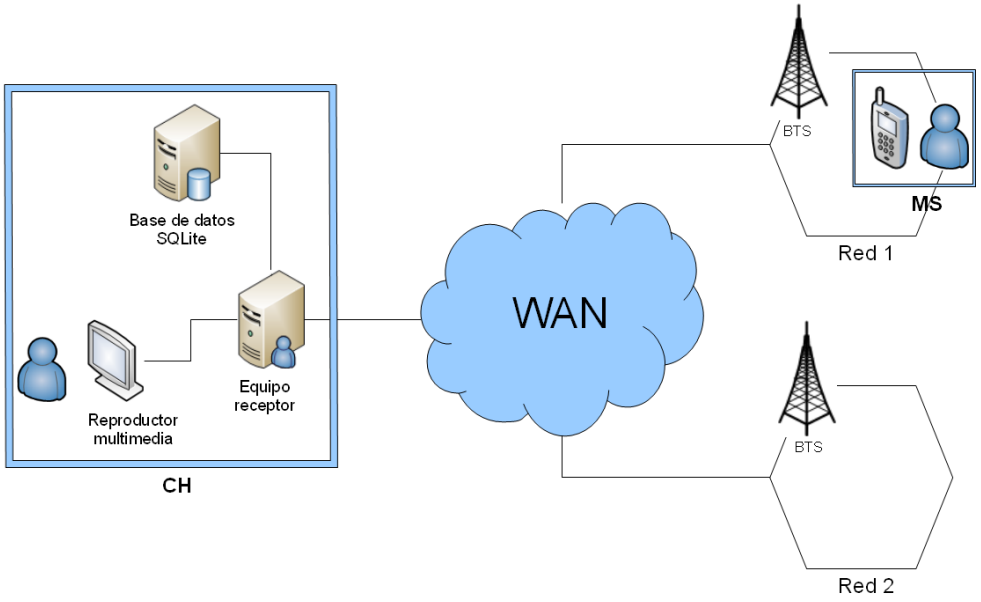

Figura 3.1: Visión simplificada del funcionamiento del sistema.

para el intercambio de información entre el terminal móvil y el equipo receptor, el teléfono comienza el envío de un flujo M-JPEG a partir de las imágenes capturadas con su cámara integrada, siempre y cuando la tenga habilitada para su uso. Para ello, MS utiliza el formato M-JPEG, que permite el envío de fragmentos de una secuencia de imágenes codificadas en formato JPEG sobre datagramas RTP, el cual permite enviar un flujo continuo de las imágenes capturadas por el terminal.

Una vez que el equipo CH conoce la dirección (*unicast* o *multicast*) a la que va dirigido dicho flujo M-JPEG, mediante la información recibida en un mensaje SDP adjunto al mensaje SIP `INVITE` enviado por MS, decide reproducir dicha secuencia de imágenes, ya que descubre que es destinatario de la misma. A partir de este momento, dicho flujo de imágenes M-JPEG/RTP es reensamblado y mostrado por pantalla en el equipo receptor.

Posteriormente, y con la sesión de datos ya establecida y en pleno funcionamiento, MS cambia de ubicación, haciéndose necesaria una migración a una nueva subred dentro de un mismo dominio, o inclusive, la migración hacía otro dominio, lo que conlleva un cambio y adquisición de una nueva IP por parte del terminal móvil (ver figura 3.3).

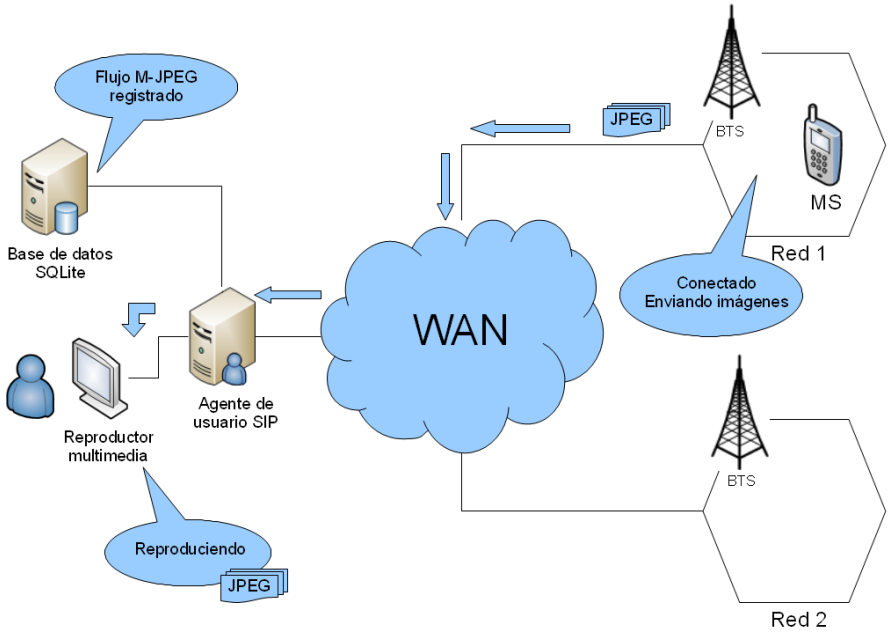

Figura 3.2: El terminal móvil comienza el envío de un flujo M-JPEG o secuencia de imágenes.

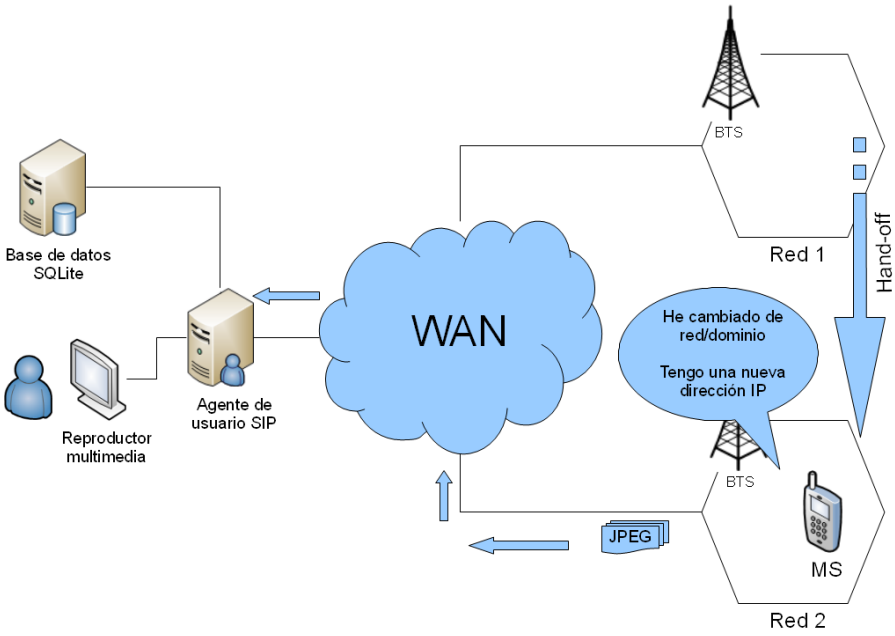

Figura 3.3: El terminal móvil cambia de ubicación.

En este contexto, podemos plantearnos dos situaciones diferentes que conlleven la adquisición de una nueva IP por parte del dispositivo móvil:

- El usuario móvil se desplaza y MS se mueve de una subred a otra, dentro de un mismo dominio administrativo, haciéndose necesario un procedimiento de *hand-off* o cambio de red intradominio.
- El usuario móvil se desplaza y MS se mueve a otra red, perteneciente a un nuevo dominio administrativo, haciéndose necesario un procedimiento de *hand-off* o cambio de red interdominio.

En ambos casos, y como ya vimos en la sección 2.3.2 y la figura 2.3, el soporte para realizar el *hand-off* o cambio de red necesario, es proporcionado mediante el protocolo SIP y el procedimiento `UPDATE`. Cuando el MS detecta el cambio de subred y/o dominio, inicia un nuevo procedimiento SIP de tipo `INVITE`, que alerta de esta situación al CH (ver figura 3.4). La implementación de este procedimiento, por parte de la aplicación práctica desarrollada en este trabajo, será detallada en capítulos posteriores.

Mediante un intercambio de mensajes SIP de tipo `INIVITE-UPDATE`, el CH actualiza la información almacenada sobre la sesión en curso con el MS, y una

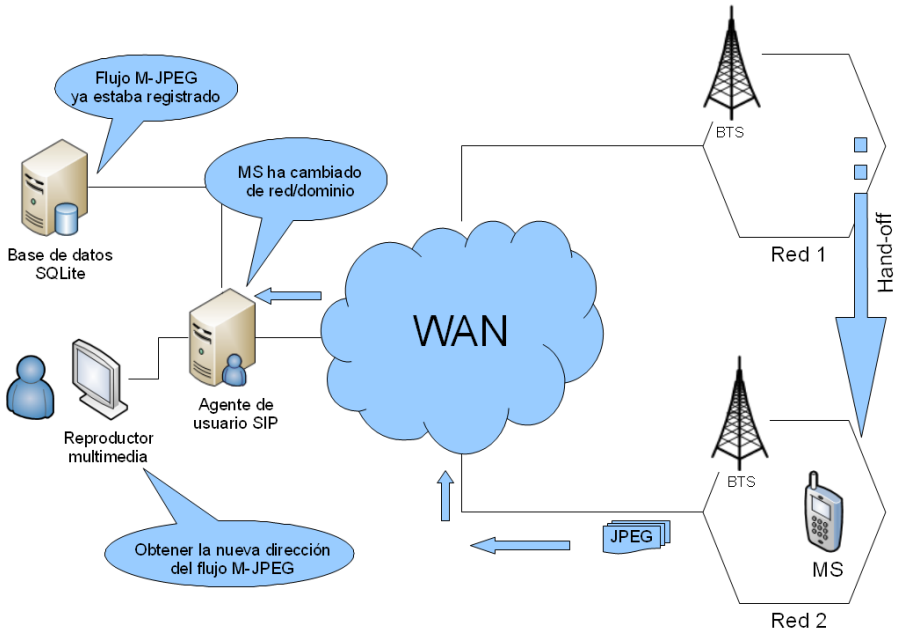

Figura 3.4: Se produce un procedimiento *hand-off* o cambio de red.

vez finalizado el proceso, el CH es capaz de continuar la reproducción del flujo M-JPEG enviado por el MS desde su nueva ubicación (ver figura 3.5). Durante este procedimiento crítico, es posible que se pierda alguna de las imágenes enviadas, o en su defecto, llegue corrompida. En estos casos, el equipo receptor deberá obviar dicho fotograma y continuar con la reproducción de las imágenes recibidas posteriores.

Por último, el usuario móvil decide detener el envío de imágenes desde su terminal, alertando al equipo receptor mediante el procedimiento SIP `BYE`, el cual pone fin a la sesión de datos establecida entre ambos extremos y libera los recursos reservados para dicha conexión.

Como ya se ha comentado anteriormente, cabe destacar que el sistema desarrollado durante este trabajo solo se encarga de implementar los extremos finales de la arquitectura para la movilidad SIP propuesta en la figura 2.2. Como puede observarse, este diseño fija su atención en el terminal móvil y el equipo receptor, por lo que, cualquier procedimiento para el registro de dispositivos en un servidor *proxy* SIP mediante `REGISTER`, queda fuera de los límites del presente documento.

En lo siguientes apartados se intentará proporcionar una introducción a las funcionalidades proporcionadas por la aplicación implementada para el terminal

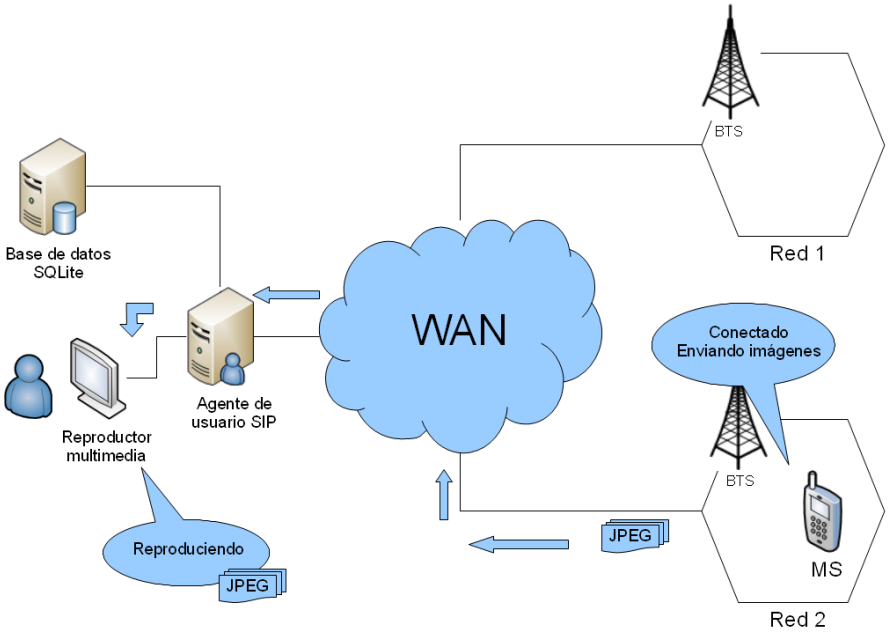

Figura 3.5: Procedimiento *hand-off* o cambio de red finalizado. Continúa la reproducción.

móvil, así como la aplicación implementada para el equipo receptor.

3.1.2. Funcionalidad de la aplicación del terminal móvil

La aplicación práctica desarrolla para el MS proporciona una serie de funcionalidades que dotan de determinado soporte al sistema.

Soporte SIP

La aplicación del terminal móvil está dotada de un agente de usuario SIP, capaz de realizar un conjunto reducido de procedimientos, de la totalidad de procedimientos descritos en la RFC 3261 [24], tales como: `REGISTER`, `INVITE`, `UPDATE`, `ACK`, `BYE`, `CANCEL`, etc.

Como ya comentamos en la sección 2.3.1, dicho agente de usuario SIP, dota a la subsistema MS de la lógica necesaria para iniciar, actualizar y finalizar sesiones SIP para el envió de información multimedia a otros equipos.

Soporte SDP

La aplicación del terminal móvil es capaz de generar información de codificación referente al flujo multimedia enviado (M-JPEG en este caso), así como otra información de conexión relevante, y prepararla en un formato adecuado para su envío al equipo receptor mediante el protocolo SDP, documentado en la RFC 4566 [26].

Soporte RTP

La aplicación del terminal móvil es capaz de gestionar conexiones RTP, para el envío de fragmentos procedentes de una secuencia de imágenes codificada en formato JPEG. Para ello, el MS implementa la RFC 3551 [27], ofreciendo soporte para la creación de datagramas del protocolo RTP.

Soporte M-JPEG

La aplicación del terminal móvil es capaz de generar paquetes M-JPEG, encargados de transportar (sobre el protocolo RTP) fragmentos procedentes de una secuencia de imágenes. Para ello, el MS implementa la RFC 2435 [28], ofreciendo soporte para la creación de datagramas M-JPEG sobre el protocolo RTP.

Soporte para procedimiento *hand-off* o cambio de red

La aplicación del terminal móvil es capaz de generar un rango de direcciones IP aleatorio y simular un cambio de red, iniciando para ello un proceso de actualización de sesión mediante el protocolo SIP con el equipo receptor.

Captura y envío de imágenes JPEG

La aplicación del terminal móvil es capaz de enviar una secuencia continua de imágenes JPEG codificas en formato M-JPEG. En una aplicación ejecutada en un teléfono real, la captura de imágenes en formato JPEG se realiza mediante la utilización de la función `take_photo(...)`, proporcionada por el módulo PyS60 `camera`. En nuestro caso, y puesto que la aplicación implementada ha sido desarrollada y probada en el emulador software S60 proporcionado por Nokia, la secuencia de imágenes enviada por el terminal estará previamente almacenada en el dispositivo, obviándose así el paso de captura anteriormente mencionado.

Interfaz gráfica de usuario

Con el objetivo de proporcionar un entorno más amigable al usuario, la aplicación implementada en la parte móvil está dotada de una interfaz gráfica de usuario. Dicha interfaz, da la posibilidad de configurar determinados parámetros de conexión de la aplicación, así como permitir la simulación de procedimientos de *hand-off* o cambios de red, entre otras funcionalidades.

Soporte multihilo

Con el objetivo de proporcionar un sistema capaz de interactuar con el usuario mientras se produce el envío información, la aplicación implementada en la parte móvil utiliza técnicas multihilo, con el fin de permitir manejar la interfaz gráfica aunque la aplicación esté ocupada, así como gestionar diferentes conexiones al mismo tiempo.

3.1.3. Funcionalidad de la aplicación del equipo receptor

La aplicación práctica desarrolla para el CH proporciona una serie de funcionalidades que dotan de determinado soporte al sistema.

Soporte SIP

La aplicación del equipo receptor está dotada de un agente de usuario SIP, capaz de interpretar un conjunto reducido de procedimientos SIP, de la totalidad de procedimientos descritos en la RFC 3261 [24], tales como: `INVITE`, `UPDATE`, `BYE`, `CANCEL`, etc., así como devolver respuestas de tipo `100`, `200` o `400`.

El agente de usuario SIP, dota a la subsistema CH de la lógica necesaria para iniciar, actualizar, cancelar y finalizar sesiones SIP para la recepción de información multimedia.

Soporte SDP

La aplicación del equipo receptor es capaz de interpretar la información de codificación referente a un flujo multimedia enviado (M-JPEG en este caso), así como otra información de conexión relevante, mediante la lectura de mensajes SDP, documentado en la RFC 4566 [26].

Soporte RTP

La aplicación del equipo receptor es capaz de gestionar conexiones RTP, para la recepción de fragmentos procedentes de una secuencia de imágenes codificada en formato JPEG. Para ello, CH implementa la RFC 3551 [27], ofreciendo soporte para la interpretación de datagramas del protocolo RTP.

Soporte M-JPEG

La aplicación del equipo receptor es capaz de interpretar el contenido de los paquetes M-JPEG recibidos, encargados de transportar (sobre el protocolo RTP) fragmentos procedentes de una secuencia de imágenes. Para ello, CH implementa la RFC 2435 [28], ofreciendo soporte para la interpretación de flujos M-JPEG sobre datagramas RTP.

Control de sesiones multimedia M-JPEG

La aplicación del equipo receptor utiliza una base de datos SQLite para el registro y control de las sesiones multimedia activas establecidas, así como el tipo de información y codificación originada por el MS (en nuestro caso, flujos M-JPEG). Concretamente, la información almacenada está basada en el identificador

del usuario móvil, el tipo de flujo multimedia emitido por el mismo, así como la dirección y puerto donde se encuentra disponible para su reproducción.

Uno de los aspectos a destacar en la utilización de SQLite, radica en que plantea un esquema que admite consultas SQL tradicionales, por lo que sería sencillo ampliar el tipo de información almacenada en la misma, si en un futuro decide realizar un diseño más complejo para el almacenamiento de información sobre el usuario conectado al sistema.

Interfaz gráfica de usuario y reproductor M-JPEG

Con el objetivo de proporcionar un entorno más amigable al usuario, la aplicación implementada en la parte receptora está dotada de una interfaz gráfica de usuario. Dicha interfaz, da la posibilidad de configurar determinados parámetros de conexión de la aplicación, así como permitir visualizar por pantalla el flujo M-JPEG recibido mediante un reproductor integrado especialmente desarrollado durante este trabajo.

En lo que respecta al procedimiento de simulación de *hand-off* o cambios de red, se mostrarán al usuario de forma totalmente trasparente.

Soporte multihilo

Con el objetivo de proporcionar un sistema capaz de interactuar con el usuario mientras se produce el envío información, la aplicación implementada en la parte receptora utiliza técnicas multihilo, con el fin de permitir manejar la interfaz gráfica aunque la aplicación esté ocupada, así como gestionar diferentes conexiones al mismo tiempo

3.2. Requisitos

Uno de los objetivos principales que persigue este proyecto, consiste en crear un sistema capaz de dar soporte para la movilidad SIP a un servicio móvil de envío de flujos M-JPEG mediante tecnología Python. Si un terminal móvil cambia de ubicación y de dirección IP, el sistema debe ser capaz de gestionar dicho procedimiento de *hand-off* o cambio de red, permitiendo que la reproducción previa en un equipo receptor continúe de forma trasparente para el usuario destinatario de dicho flujo.

En este sentido, ha sido necesario definir una serie de requerimientos funciona-

les y especificaciones de diseño, que permitan desplegar e implementar el sistema requerido. Por esta razón, a continuación se enumeran los principales requisitos del diseño propuesto en este trabajo.

3.2.1. Requisitos funcionales

1. En el sistema participarán dos tipos de entidades o usuarios. Los encargados de la reproducción y visualización de los datos multimedia en un equipo receptor, y los usuarios móviles, responsables del envío de las secuencias de imágenes o flujos M-JPEG.

2. El sistema se basará en una arquitectura tipo cliente-servidor, y estará sustentado respectivamente por dos entidades que asumirán el control del envío de datos (por parte del terminal móvil), y su recepción y visualización (en una máquina IP) a través de la red.

3. El sistema basará su funcionamiento en el envío de datos multimedia, preferiblemente imágenes o vídeo desde el terminal móvil a otro sistema o máquina que pueda ser enrutado mediante una dirección IP. La visualización de las imágenes se llevará acabo mediante algún tipo de reproductor multimedia en el equipo receptor. En este aspecto, se sugiere la utilización de VLC, un reproductor y *framework* multimedia del grupo VideoLAN, proyecto de software libre distribuido bajo la licencia GPL; aunque no se descarta la creación de un sencillo reproductor basado en el lenguaje de programación Python y PIL (*Python Imaging Library*), una librería externa para dicho lenguaje, que añade soporte para la apertura, manipulación y guardado de imágenes en diferentes formatos, soportando versiones Python desde la 2.2 a la 2.6.

4. El sistema deberá proporcionar sendas interfaces de usuario en ambos extremos, para facilitar el manejo de la aplicación. En la parte móvil, dicha interfaz estará basada en el uso de módulos PyS60 disponibles para terminales Symbian S60, mientras que en la aplicación del equipo receptor se basará en el conjunto de módulos Python de la librería estándar, así como en los módulos externos que fueran necesarios.

5. El sistema multimedia desarrollado debe permitir la movilidad de los usuarios en Internet mediante el protocolo SIP. Dicho procedimiento de *hand-off* o cambio de red estará basado en el borrador del IETF de 2001 “Soporte para la movilidad multimedia con SIP” [23]. En este sentido, se tendrán en cuenta dos posibles casos o situaciones:

   a) El sistema diseñado debe ofrecer soporte de movilidad a un terminal móvil Symbian S60 mediante el protocolo SIP, permitiendo el cambio y conexión a distintas subredes pertenecientes a un mismo dominio.
   b) El sistema diseñado debe ofrecer soporte de movilidad a un terminal móvil Symbian S60 mediante el protocolo SIP, permitiendo el cambio y conexión a distintas redes pertenecientes a un dominio distinto al actual.

6. La aplicación práctica desarrollada para el terminal móvil debe permitir simular un cambio de red en cualquier momento, y de forma independiente al envío de la secuencia de imágenes o flujo M-JPEG transmitido. Dicho cambio se notificará al equipo receptor mediante el correcto intercambio de mensajes SIP.

7. La aplicación práctica desarrollada para el equipo receptor deberá permitir la gestión y control, mediante el protocolo SIP, de los procedimientos de cambio de red iniciados en el terminal móvil.

3.2.2. Requisitos técnicos

1. El sistema práctico implementado deberá desarrollarse exclusivamente mediante el lenguaje de programación Python. El software implementado para el dispositivo móvil hará uso de los módulos PyS60 (versión 1.4.5), específicos para la serie 60 de terminales celulares dotados con Symbian OS. El software implementado para el equipo receptor será desarrollado mediante Python 2.x y el conjunto de librerías estándar proporcionadas por dicho lenguaje, así como el conjunto externo de librerías Python necesario.

2. La implementación de la aplicación móvil práctica se diseñará específicamente para su correcto funcionamiento en terminales S60 (versión 3.0) 3ª Edición *Feature Pack* 1, dotados de sistema operativo Symbian OS (versión 9.2).

3. La aplicación móvil implementada deberá ser compatible con un terminal Symbian S60 3ª Edición *Feature Pack* 1, o funcionar correctamente en el emulador software S60 proporcionado por el SDK *Developers Tools* de Nokia.

4. El sistema deberá garantizar, de forma robusta, el establecimiento de sesiones multimedia mediante el protocolo SIP, así como la actualización de los parámetros de la conexión, y en última instancia, la finalización de la mismas. Para ello se implementará parcialmente la RFC 3261 [24].

5. El envío de vídeo o imágenes se realizará sobre el protocolo RTP y UDP. Para ello se implementará la RFC 3551 [27].

6. Si finalmente se decide enviar una secuencia de imágenes, se utilizará un formato M-JPEG sobre el protocolo RTP. En dicho caso se implementará la RFC 2435 [28].

7. El anuncio de los nuevos flujos o secuencias de imágenes enviados desde el terminal móvil al equipo receptor desde el terminal móvil, deberán ser anunciados mediante un mensaje SDP. Para ello se implementará parcialmente la RFC 4566[26].

8. El sistema deberá realizar una gestión de usuarios y flujos de imágenes eficiente y escalable, por si en un futuro se desea ampliar el tipo de información almacenada en el mismo. Para ello se propone el uso de una base de datos ligera como SQLite [12], la cual sigue la especificación DB-API 2.0 [13].

9. Tanto aplicación móvil, como aplicación en el equipo receptor, deberán proporcionar un soporte multihilo, tanto en el establecimiento y gestión de las conexiones, así como para evitar bloqueos a la hora de manejar su interfaz gráfica de usuario.

3.3. Arquitectura

Esta sección está dedicada a presentar una visión introductoria de los módulos implementados, para el correcto despliegue del sistema propuesto a lo largo de los anteriores epígrafes.

El objetivo es ofrecer una estructura física de los módulos Python desarrollados, tanto para la aplicación del terminal móvil, como para la aplicación del equipo receptor, así como las entidades lógicas que representan y las funciones que realizan.

Por último, también se detalla la torre de protocolos utilizada por el sistema, tanto en el plano de usuario (para el envío de información), como en el plano de control (para el establecimiento y actualización de sesiones multimedia).

3.3.1. Despliegue

En la figura 3.6 podemos observar el diagrama de despliegue y los componentes del sistema desarrollado. Este diagrama representa la jerarquía o estructura física

real de los módulos Python implementados durante este trabajo, así como su interacción entre ellos y con los usuarios finales del sistema mediante sendas interfaces gráficas de usuario.

Como puede observarse en la figura 3.6, la aplicación móvil está gobernada por un módulo principal denominado `MS`, el cual se encarga de manejar los eventos producidos en la interfaz gráfica de usuario, provenientes de las acciones realizadas por el individuo que maneja el terminal móvil. Además, gracias a que `MS` cuenta con soporte multihilo, es capaz de realizar otras tareas en paralelo, como el establecimiento y actualización de sesiones multimedia, o el envío de flujos M-JPEG sobre el protocolo RTP.

Cabe destacar que, tanto la aplicación implementada en la parte del terminal móvil, como la aplicación implementada en el equipo receptor, utilizan una serie de módulos especiales: `SIPmessages`, `SDPmessages`, `RTPpackets`, `MJPEGpackets`. Estos módulos comunes ofrecen funcionalidades generales, que pueden ser utilizadas por ambos extremos del sistema. Dichas funcionalidades, así como los detalles técnicos sobre la implementación de dichos módulos, serán explicadas con mayor detalle en el capítulo 6 del presente documento.

Por otra parte, y como ya se introdujo en la sección 3.1.2 del presente documento, `MS` utiliza los módulos comunes del sistema para ofrecer las siguientes características:

- La aplicación `MS`, apoyándose en las funcionalidades ofrecidas por el módulo `SIPmessages`, es capaz de proporcionar soporte para el protocolo SIP, así como proporcionar un agente de usuario SIP con la lógica suficiente para establecer, actualizar y finalizar sesiones multimedia con el equipo receptor.

- La aplicación `MS`, apoyándose en las funcionalidades ofrecidas por el módulo `SDPmessages`, es capaz de proporcionar soporte para el protocolo SDP, el cual permite el envío de mensajes con información sobre la codificación del flujo multimedia enviado por el terminal móvil al equipo receptor.

- La aplicación `MS`, apoyándose en las funcionalidades ofrecidas por el módulo `RTPpackets`, es capaz de proporcionar soporte para el protocolo RTP, el cual permite el envío de paquetes con fragmentos de información pertenecientes a un flujo multimedia sobre un *socket* UDP.

- La aplicación `MS`, apoyándose en las funcionalidades ofrecidas por el módulo `MJPEGpackets`, es capaz de proporcionar soporte M-JPEG, el cual permite el envío de paquetes con fragmentos de imágenes pertenecientes a una secuencia completa sobre el protocolo RTP.

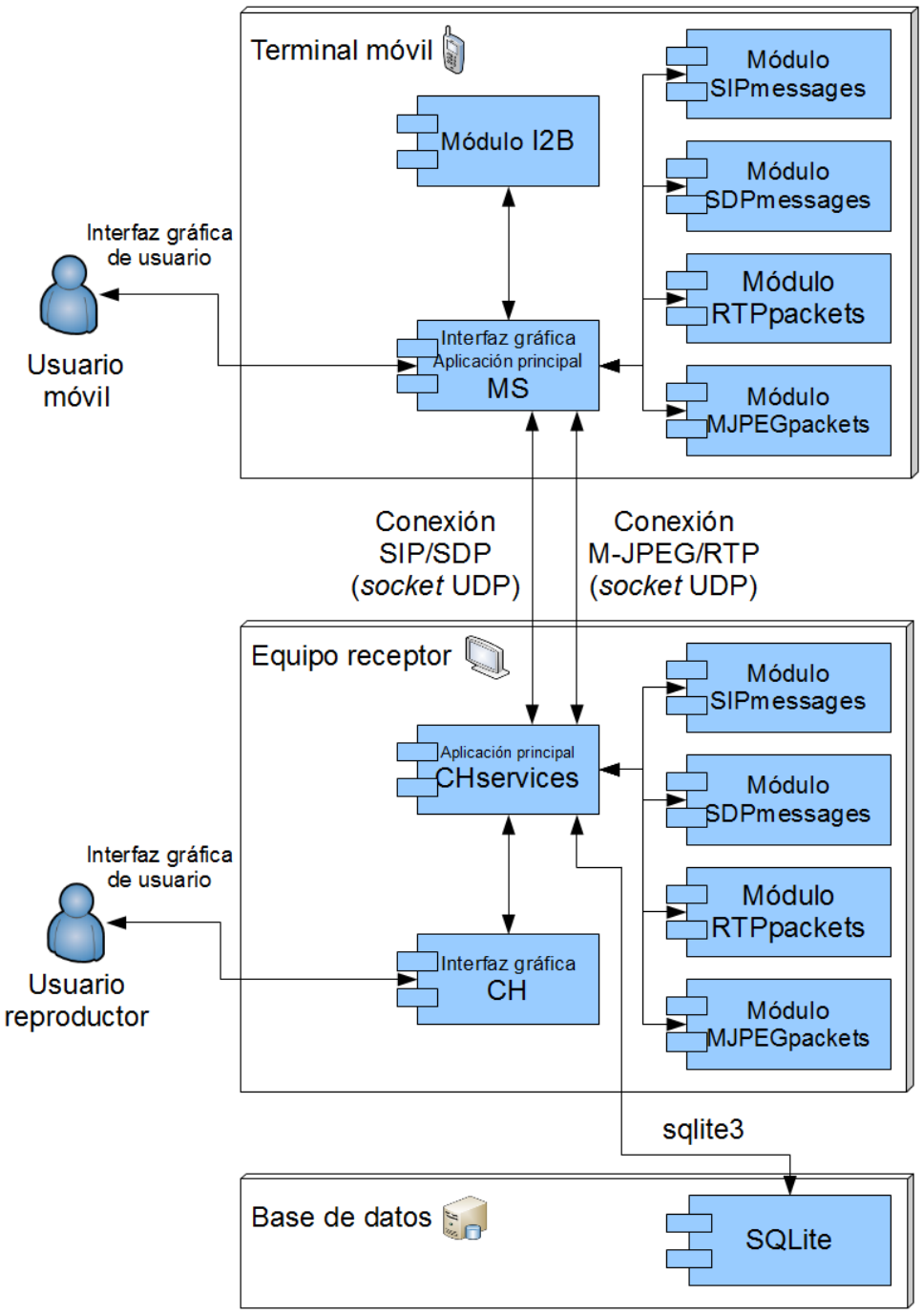

Figura 3.6: Diagrama de despliegue y componentes del sistema implementado.

Además, el módulo principal `MS` utiliza otros módulos como `I2B` a modo de utilidades complementarias de la aplicación móvil.

Por su parte, y como puede observarse en la figura 3.6, la aplicación receptora está divida en dos módulos diferenciados. El módulo denominado `CH` se encarga de manejar los eventos producidos en la interfaz gráfica de usuario, provenientes de las acciones realizadas por el individuo que maneja el el equipo receptor o reproductor. Por otro lado, el módulo denominado `CHservices` implementa la lógica de la aplicación y las funcionalidades requeridas por el diseño. Además, gracias a que `CHservices` cuenta con soporte multihilo, es capaz de realizar varias tareas en paralelo, como el establecimiento y actualización de sesiones multimedia, o la recepción y reproducción de flujos M-JPEG por pantalla.

Por otra parte, y como ya se introdujo en la sección 3.1.3 del presente documento, `CHservices` utiliza los módulos comunes del sistema para ofrecer las siguientes características:

- La aplicación `CHservices`, apoyándose en las funcionalidades ofrecidas por el módulo `SIPmessages`, es capaz de proporcionar soporte para el protocolo SIP, así como proporcionar un agente de usuario SIP con la lógica suficiente para establecer, actualizar y finalizar sesiones multimedia con un terminal móvil.

- La aplicación `CHservices`, apoyándose en las funcionalidades ofrecidas por el módulo `SDPmessages`, es capaz de proporcionar soporte para el protocolo SDP, el cual permite la interpretación de los parámetros enviados en un mensaje SDP, con información sobre la codificación del flujo multimedia enviado por el terminal móvil al equipo receptor.

- La aplicación `CHservices`, apoyándose en las funcionalidades ofrecidas por el módulo `RTPpackets`, es capaz de proporcionar soporte para el protocolo RTP, el cual permite la recepción y reensamblado de los paquetes RTP recibidos sobre un *socket* UDP en el equipo receptor, los cuales contienen fragmentos de información pertenecientes a un flujo multimedia enviado por el terminal móvil.

- La aplicación `CHservices`, apoyándose en las funcionalidades ofrecidas por el módulo `MJPEGpackets`, es capaz de proporcionar soporte M-JPEG para la interpretación de los paquetes recibidos en el equipo receptor, los cuales contienen fragmentos de imágenes pertenecientes a una secuencia completa.

Por último, `CHservices` se ocupa de mantener y actualizar la información referente a la sesiones multimedia activas, así como el tipo de datos multimedia

reproducido en dicha sesión. Para ello utiliza el módulo Python estándar `sqlite3`, el cual permite gestionar una base de datos ligera SQLite.

3.3.2. Entidades lógicas

Si bien en la sección 3.3.1, hemos presentado la estructura de los módulos Python desarrollados para la implementación del diseño propuesto, en este apartado vamos a detallar las entidades lógicas que participan en el sistema y los módulos que las representan.

El sistema posee varios tipos de entidades lógicas diferenciadas, encargadas de gestionar cada uno de los aspectos del esquema planteado. En los siguientes epígrafes pasamos a enumerarlas y comentar brevemente el cometido de cada una de ellas, posponiendo para capítulos sucesivos los detalles técnicos involucrados en su diseño y funcionamiento.

Interfaz gráfica de usuario del MS

Representa el punto de entrada de la aplicación móvil, y permite al usuario decidir en que momento comienza, se actualiza o finaliza una sesión de envío de secuencias de imágenes M-JPEG.

En el caso del terminal móvil, el código fuente, que implementa la lógica de la interfaz desarrollada para la aplicación práctica, está ubicado en el fichero `MS.py`. Este es el módulo principal del sistema y el punto de inicio de la aplicación, ya que se encarga de inicializar diversos parámetros del teléfono, como la dirección IP del mismo, o fijar la dirección IP del equipo receptor a quien se enviará la secuencia M-JPEG.

Gracias al módulo PyS60 `appuifw`, dicha interfaz cuenta con varios menús que permiten configurar la dirección destino del equipo receptor, así como tener un control "en vivo" del inicio y finalización de sesiones SIP, y permitir realizar simulaciones de procedimientos de cambio de red o *hand-off* en cualquier momento que el usuario desee.

Interfaz gráfica de usuario del CH

Representa el punto de entrada de la aplicación en el equipo receptor, y permite al usuario decidir en que momento debe activarse el equipo reproductor para permanecer a la escucha de nuevos mensajes SIP, procedentes del terminal, que indiquen el inicio, actualización por *hand-off* o finalización, de sesiones

multimedia para la recepción de flujos M-JPEG.

Así mismo, la interfaz también permite la introducción de diversos parámetros del equipo receptor, como la dirección IP o el puerto donde escuchar nuevas conexiones SIP para el establecimiento o actualización de sesiones multimedia.

En el caso del equipo receptor, el código fuente, que implementa la lógica de la interfaz desarrollada para la aplicación práctica, está ubicado en el fichero `CH.py`, la cual permite instanciar objetos de la clase `CH` para su posterior manipulación.

Agente de usuario SIP del MS

Implementa la lógica necesaria para establecer sesiones SIP mediante mensajes `INVITE`, actualizarlas mediante mensajes SIP `UPDATE`, y finalizarlas mediante mensajes SIP `BYE` con el equipo CH. Este servicio lo presta un hilo denominado `start`, perteneciente a la aplicación principal `MS`.

Así mismo, el agente de usuario SIP de MS es capaz de lanzar hilos de ejecución para el envío de secuencias de imágenes M-JPEG, así como decidir cuando es preciso realizar un procedimiento de *hand-off*.

En el caso del terminal móvil, el código fuente, que implementa la lógica del agente de usuario SIP para la aplicación práctica desarrollada, está ubicado en los ficheros `MS.py`, `SIPmessages.py` y `SDPmessages.py`.

Agente de usuario SIP del CH

Implementa la lógica necesaria para permitir establecer sesiones SIP mediante mensajes `INVITE`, actualizarlas mediante mensajes SIP `UPDATE`, y finalizarlas mediante mensajes `BYE` con el terminal MS, así como generar respuestas SIP de tipo 200 (`Ok`) y 400 (`Bad Request`). Este servicio lo presta un hilo denominado `run`, el cual es invocado mediante la función `threading.start()`, la cual actúa sobre los objetos de la clase `CHservices`, que a su vez deriva de la clase `threading.Thread`.

Así mismo, el agente de usuario SIP de CH es capaz de lanzar hilos de ejecución para la recepción y reproducción de secuencias de imágenes M-JPEG, así como decidir cuándo es preciso realizar un procedimiento de *hand-off*.

En el caso del equipo receptor, el código fuente del agente de usuario SIP para la aplicación práctica desarrollada, está ubicado en los ficheros `CHservices.py`, `SIPmessages.py` y `SDPmessages.py`.

Generadores de flujos M-JPEG

Son hilos de ejecución, lanzados por la aplicación del MS, capaces de enviar secuencias de imágenes M-JPEG sobre el protocolo RTP para una dirección y puerto determinados. Este servicio lo presta un hilo denominado `sendFrames` perteneciente a la aplicación principal `MS` del terminal móvil.

El código fuente, que implementa la lógica de estos generadores de flujos M-JPEG para la aplicación práctica desarrollada, está ubicado en los fichero `MS.py`, `RTPpackets.py` y `MJPEGpackets.py`.

Consumidores de flujos M-JPEG

Son hilos de ejecución, lanzados por la aplicación del CH, capaces de reproducir por pantalla secuencias de imágenes M-JPEG sobre el protocolo RTP para una dirección y puerto determinados. Este servicio lo presta un hilo denominado `receiveFrames` perteneciente a la aplicación principal `CHservices` del equipo receptor.

El código fuente, que implementa la lógica de estos consumidores de flujos M-JPEG para la aplicación práctica desarrollada, está ubicado en los ficheros `CHservices.py`, `RTPpackets.py` y `MJPEGpackets.py`.

Gestor de sesiones multimedia M-JPEG

La gestión de las sesiones de usuarios establecidas con el equipo receptor, así como los parámetros de conexión de los flujos M-JPEG recibidos sobre UDP/RTP, son registrados en una base de datos ligera SQLite, permitiéndose su almacenamiento, consulta y actualización mediante lenguaje PL/SQL convencional.

En el caso del equipo receptor, el código fuente que implementa la lógica para la gestión y control de las sesiones de datos M-JPEG establecidas en la aplicación práctica desarrollada, está ubicado en el fichero `CHservices.py`.

Otras entidades

1. Proveedor de servicios y suministrador de direcciones IP: En el caso de un terminal móvil real, la IP de dicho dispositivo será suministrada por un proveedor de servicios autorizado, responsable de proporcionar acceso, a las entidades involucradas en la aplicación, a un medio que permita la

comunicación entre diversos usuarios. En este caso, el terminal móvil debería utilizar las funciones proporcionadas por el módulo PyS60 `socket` para acceder a su punto de acceso a la red. Entre estas funciones podemos destacar: `select_access_point()` (abriría un diálogo de selección con una lista de los puntos de acceso disponibles, permitiendo al usuario seleccionar uno y devolviendo la ID del mismo), `access_point(apid)` (crearía un objeto `access_point` a partir de su ID), `set default access point(apo)` (configura el punto de acceso por defecto) y `access points()` (lista las ID de los puntos de acceso disponibles).

En el caso concreto de nuestra aplicación, el entorno de pruebas se reducirá a *localhost*. En este contexto, el equipo receptor tendrá una IP fija asignada de forma predeterminada desde un principio (en el rango de *loopback*), y el terminal móvil adquirirá una al encenderse de manera aleatoria (en el rango de *loopback* y siempre diferente a la que posea el equipo receptor). Posteriormente, y mediante la función `hand_off()` (ubicada en el módulo principal `MS`) el terminal móvil adquirirá una nueva dirección IP (en el rango *localhost*) para simular un procedimiento *hand-off* o de cambio de red.

2. Usuarios finales: Son los encargados de interaccionar con los servicios proporcionados por la aplicación. De este modo, un usuario móvil decidirá cuando comenzar o finalizar el envío de una secuencia de datos multimedia, además de controlar el inicio del intercambio de mensajes SIP por medio de un agente de usuario instalado en su dispositivo móvil.

   Por su parte, el usuario del equipo receptor será capaz de decidir el momento preciso en el que comienza el servicio, dejando el sistema preparado para comenzar la reproducción de las imágenes enviadas desde el teléfono móvil (o el emulador software S60 en su defecto).

3.3.3. Clases y módulos

Las entidades lógicas descritas en la sección 3.3.2, así como el diagrama de despliegue de la figura 3.6 documentado en la sección 3.3.1, se traducen a la arquitectura física mostrada en el diagrama de clases y módulos en la figura 3.7.

En el diagrama de la figura 3.7 pueden observarse los atributos y constantes, así como las funciones implementadas para cada uno de los componentes del sistema.

Presentar esta figura en la sección actual posee un mero carácter introductorio, ya que tanto los componentes de la aplicación `MS` desarrollada para el terminal móvil, los componentes de la aplicación `CH` desarrollada para el equipo receptor,

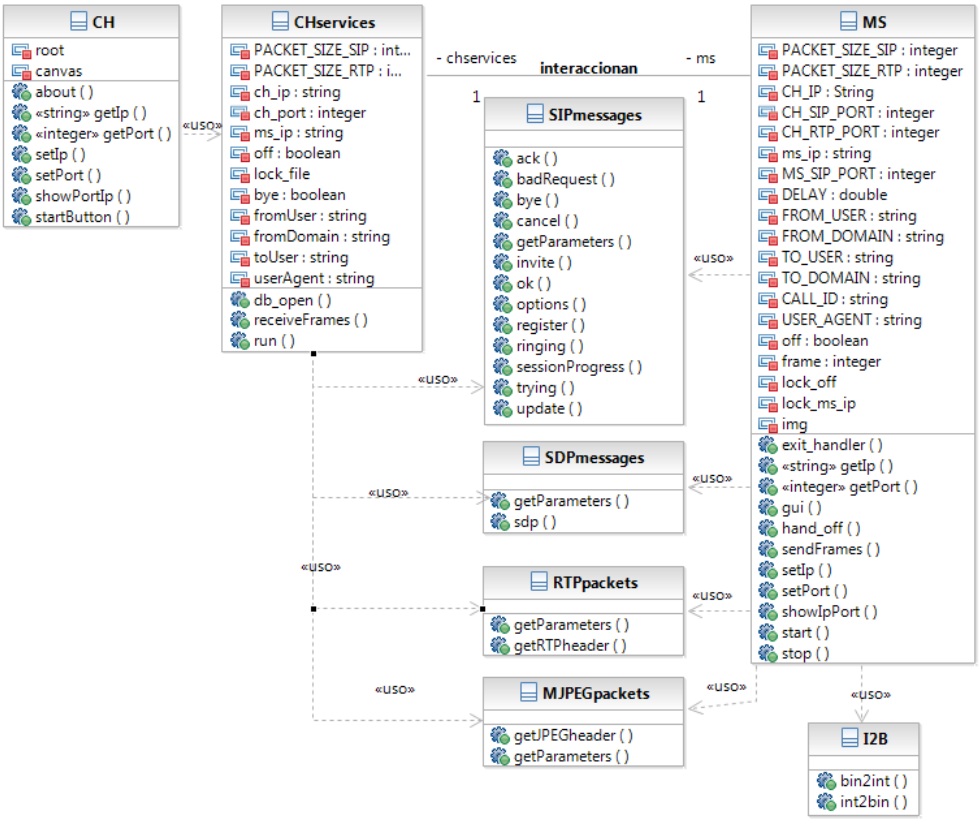

Figura 3.7: Diagrama de clases y módulos del sistema total.

así como los módulos comunes utilizados por ambas, serán explicados con mayor detalle a lo largo de los capítulos 4, 5 y 6, respectivamente.

3.3.4. Arquitectura de protocolos de comunicaciones

Con el objetivo de dar soporte a las funcionalidades requeridas por el sistema, el diseño desarrollado implementa una torre de protocolos como la mostrada en la figura 3.8. Dicha figura esquematiza, de forma generalizada, la estructura y protocolos de comunicaciones utilizados en un diseño general dentro de un marco de arquitectura para la movilidad en Internet.

En el caso concreto de la aplicación desarrollada para este trabajo, el entorno de desarrollo y de pruebas se restringe al propio equipo y el emulador software S60 proporcionado por el SDK de *Nokia Developers Tool* (ver figura 3.9).

Plano de datos del usuario

Como puede observarse de forma simplificada en la figura 3.9, para enviar una secuencia de imágenes M-JPEG al equipo receptor (CH), el terminal móvil (MS) se apoya en el establecimiento de un *socket* UDP no orientado a conexión, para el envío de datos sobre RTP. En nuestro caso concreto, dicho conjunto de datos representan una colección de fragmentos de una imagen o *frame*, pertenecientes a una secuencia de fotogramas tomadas por la cámara del terminal móvil, o en su defecto, previamente almacenadas en la memoria del emulador software.

A su vez, cada uno de los fragmentos enviados va precedido de una cabecera M-JPEG que indica el *offset* o desplazamiento de dicho fragmento dentro del *frame* o imagen enviada, así como otros parámetros de interés, como por ejemplo la resolución de la misma.

Una vez establecida la sesión multimedia SIP entre ambos equipos, el terminal móvil procede al envío de dicha secuencia imágenes, las cuales serán recibidas y debidamente visualizadas en un reproductor multimedia.

Una vez los fotogramas sean recibidos en el CH, la aplicación del equipo receptor lanzará un hilo que se encargue de reensamblar los fragmentos obtenidos, y mostrarlos por la pantalla del reproductor multimedia predeterminado.

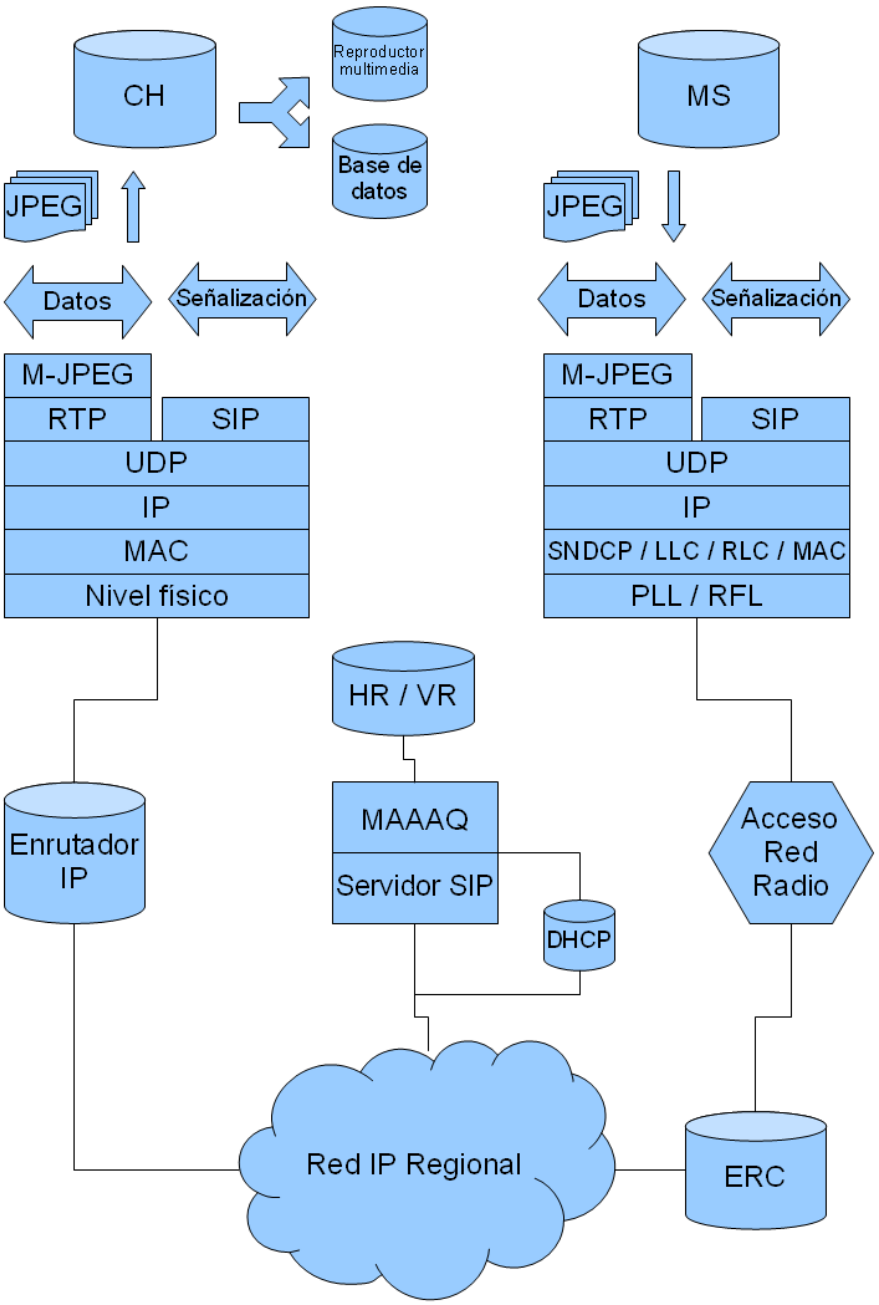

Figura 3.8: Arquitectura de protocolos de comunicaciones utilizada en una aplicación real.

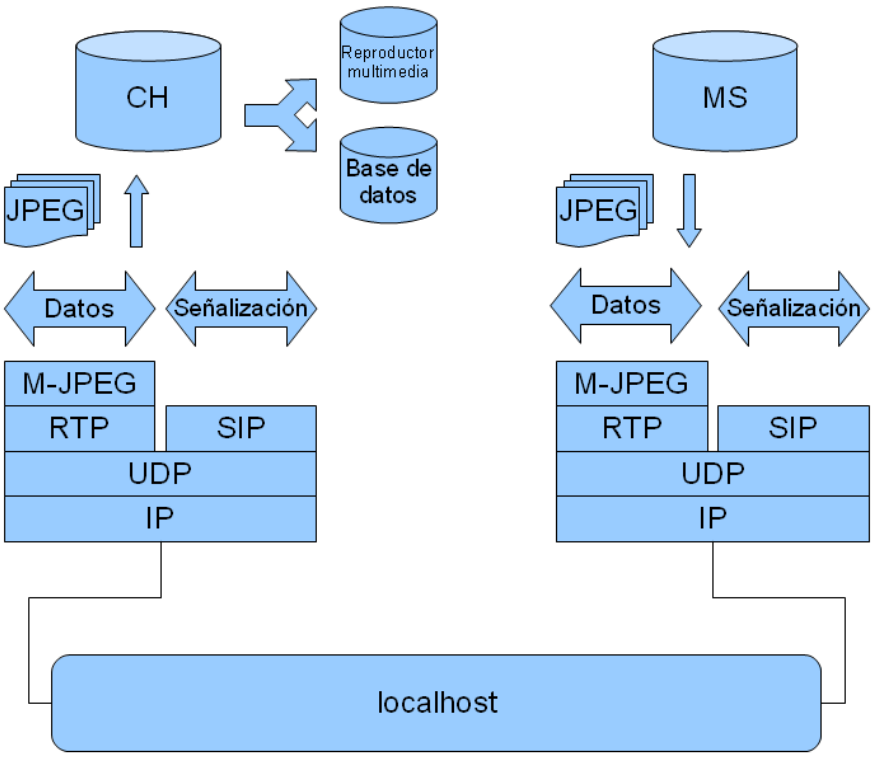

Figura 3.9: Visión simplificada de la arquitectura de protocolos utilizada en la aplicación desarrollada.

Plano de control y señalización

En el plano de control, el protocolo SIP es el encargado de gestionar y señalizar el establecimiento, curso y finalización de una sesión multimedia para el envío de datos entre el MS y el equipo CH.

Como ya comentamos en apartados anteriores, el protocolo SIP será el encargado de proveer a un terminal móvil de capacidades necesarias para permitir simular un procedimiento de *hand-off* o cambio de red en un instante aleatorio, proporcionando así el soporte para la movilidad en Internet requerido por el sistema.

Por otra parte, la aplicación receptora del flujo de datos multimedia, será la encargada de controlar las conexiones activas y los flujos M-JPEG recibidos, mediante su correcta gestión en una base de datos, con el objetivo controlar en todo momento la ubicación del usuario y el estado del flujo M-JPEG enviado desde el terminal móvil.

3.4. Caso de Uso

La funcionalidad principal del diseño propuesto en este trabajo, se basa en dotar, al usuario, de aplicaciones móviles desarrolladas con Python y PyS60, que permitan la movilidad entre redes IP mediante la utilización del protocolo SIP.

El software desarrollado en el presente proyecto permite además el envío de un flujo de imágenes, codificadas en formato JPEG, procedentes de la cámara de un dispositivo móvil, así como su visualización en un reproductor, implementado a tal efecto en un equipo IP receptor.

Teniendo en cuenta estas dos consideraciones básicas, podemos citar algunos casos de uso posibles.

3.4.1. Monitorización de pacientes con enfermedades neurodegenerativas

En pacientes que presenten cuadros de Alzheimer o demencia senil, así como otros trastornos de memoria, en determinados casos podría ser de utilidad visualizar donde se encuentra el sujeto. En estos casos, podría implementarse un sistema sencillo y económico basado en el diseño planteado en este trabajo, que reporte imágenes a través de un terminal móvil que el paciente llevará colgado, ya sea mediante una correa o fijado a una de sus prendas.

Un diseño más avanzado, y mediante el uso combinado de otros módulos PyS60 como `location` (el cual ofrece la posibilidad de acceder a diversos parámetros GSM) o `positioning` (con el que podemos acceder a las capacidades GPS del terminal), permitirían crear un sistema más completo para la localización de este tipo de pacientes en el caso de que se desorienten.

3.4.2. Sistema de difusión de imágenes

Aunque en un principio el sistema está diseñado para soportar y reproducir un flujo *unicast*, unas ligeras modificaciones en el mensaje SDP enviado por el terminal móvil, así como en su algoritmo de envío sobre M-JPEG y RTP/UDP, podrían convertirlo en un sistema de difusión *multicast*. De este modo, un usuario dispondría de un sencillo método para enviar simultáneamente imágenes desde la cámara de su teléfono a otros usuarios distribuidos geográficamente. El sistema permitiría visualizar y reproducir dichas imágenes desde cualquier lugar conectado a la red IP, así como ver la misma secuencia simultáneamente desde diferentes equipos.

Un sistema sencillo de difusión M-JPEG, que aunque puede ser superado por otro tipo de codificaciones de vídeo más eficientes, resulta económico de implementar.

3.4.3. Sistema de seguridad en transportes

Una de las ventajas que presenta el *codec* M-JPEG es la facilidad para desarrollar un mecanismo en el visor (mediante el procesado de las imágenes JPEG) para la detección de movimiento en directo.

Desde este punto de vista, el sistema desarrollado también podría servir como base de un sistema sencillo de seguridad móvil. Un centro de control recibiría las imágenes enviadas desde una serie de dispositivos móviles, ubicados en diferentes medios de transporte (autobuses, furgones blindados, etc.), ya sea para su grabación o posterior aplicación de un procesado (para detectar movimiento, etc.).

Capítulo 4

Arquitectura de la aplicación del terminal móvil

En el presente capítulo, se intentará profundizar en los detalles del software implementado para el terminal móvil mediante el lenguaje de programación Python y conjunto de librerías PyS60, especialmente desarrolladas para su uso en terminales Symbian de la serie S60.

Como ya se ha comentado en secciones anteriores, el presente proyecto centra su atención en dar soporte para la movilidad en Internet, mediante el protocolo SIP, a una aplicación móvil que se encuentre enviando un flujo de datos multimedia continuado.

En los siguientes epígrafes se explicará detalladamente la estructura, interfaces y el funcionamiento de la aplicación propuesta para la parte móvil del sistema. Para ello, a lo largo de este capítulo se describirán una serie de características técnicas del diseño implementado, las cuales permitirán complementar la visión general sobre el funcionamiento del sistema, ofrecida hasta el momento por el capítulo 3 del presente documento.

4.1. Estructura modular

Durante el presente capítulo centraremos nuestra atención exclusivamente en la parte móvil del sistema implementado (ver figura 4.1) del diagrama completo de clases y módulos (ver figura 3.7 de la sección 3.3.3).

La jerarquía de archivos, que corresponde físicamente al diagrama de la figura 4.1, se ilustra en la imagen 4.2. Ambos esquemas proporcionan una visión

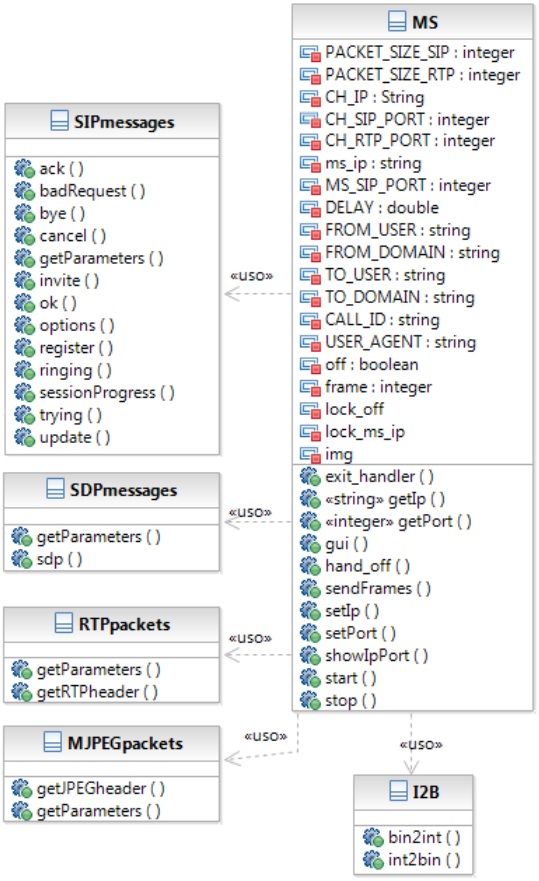

Figura 4.1: Diagrama de clases y módulos de la aplicación móvil.

completa sobre la estructuración modular de la aplicación desarrollada para el terminal móvil.

La aplicación del terminal móvil cuenta con un fichero principal denominado `MS.py`, que representa el módulo principal del sistema, ya que se encarga de desempeñar las siguientes tareas:

1. Cargar y controlar la interfaz gráfica de usuario de la aplicación desarrollada para el terminal móvil S60.

2. Obtener la dirección IP inicial del dispositivo.

3. Establecer sesiones multimedia con un equipo receptor mediante el protocolo SIP y SDP.

4. Enviar flujos o secuencias de imágenes M-JPEG sobre UDP/RTP.

5. Simular un procedimiento de *hand-off* o cambio de red del terminal móvil, y notificar dicha actualización al equipo receptor mediante el protocolo SIP y SDP.

6. Finalizar sesiones multimedia con un equipo receptor mediante el protocolo SIP.

Cuando el usuario enciende el teléfono (o en su defecto, el emulador software proporcionado por el SDK de *Nokia Developers Tool*), el intérprete Python para S60 instalado en dicho terminal será el encargado de ejecutar la aplicación `MS`, lo que inicializará el sistema mostrando una pantalla de bienvenida, para cargar posteriormente la interfaz gráfica de usuario que permitirá manejar dicha aplicación móvil, y de la que hablaremos la sección 4.3.

La aplicación `MS.py` está desarrollada mediante un estilo de programación estructurada, y utiliza las funciones proporcionadas por una serie de módulos comunes del sistema, implementados en este trabajo, para lo cual, importa los siguientes archivos Python:

- `SIPmessages.py`: Módulo común del sistema implementado en base a la RFC 3261 [24] y el *IETF Draft Supporting Mobility for Multimedia with SIP*. Contiene un conjunto de funciones encargadas de generar e interpretar un grupo de mensajes SIP, para ser utilizados en el establecimiento y finalización de sesiones multimedia, así como en otros procedimientos tales como actualización de la información de la conexión, asentimientos, etc.

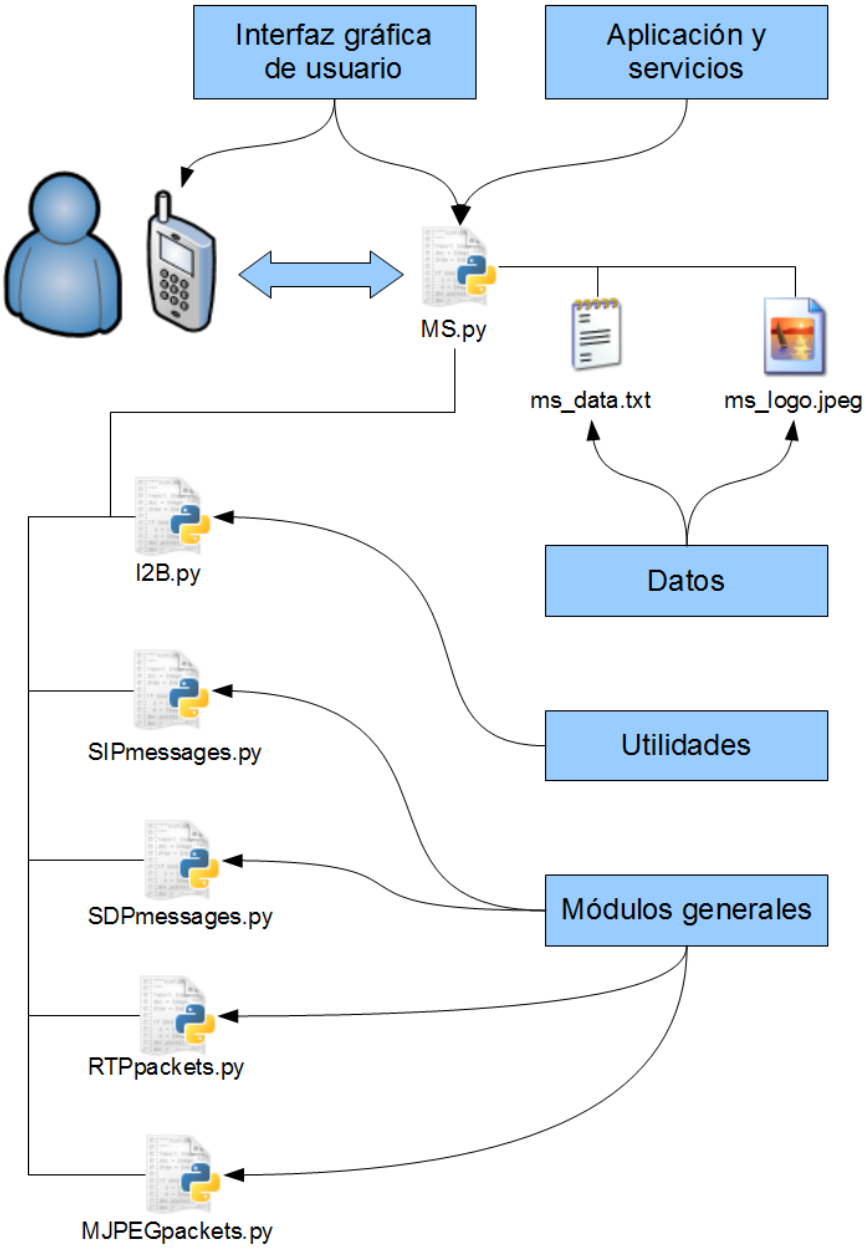

Figura 4.2: Jerarquía de archivos y módulos PyS60 desarrollados para la aplicación móvil.

- `SDPmessages.py`: Módulo común del sistema implementado en base a la RFC 4566 [26]. Contiene un conjunto de funciones encargadas de generar e interpretar un grupo de mensajes SDP, para ser utilizados en el establecimiento y actualización de sesiones multimedia.

- `RTPpackets.py`: Módulo común del sistema implementado en base a la RFC 3551 [27]. Contiene un conjunto de funciones encargadas de generar e interpretar las cabeceras de los paquetes RTP, para ser utilizados como contenedores de fragmentos de una secuencia o flujo de imágenes en formato M-JPEG.

- `MJPEGpackets.py`: Módulo común del sistema implementado en base a la RFC 2435 [28]. Contiene un conjunto de funciones encargadas de generar e interpretar las cabeceras de los paquetes M-JPEG, que precederán a los fragmentos de una secuencia o flujo de imágenes en formato M-JPEG.

Los módulos comunes del sistema permiten establecer comunicaciones SIP y enviar flujos M-JPEG sobre RTP desde el MS al CH, y están pensados para ser reutilizados fácilmente por cualquier otro programa que desarrollemos, siendo reaprovechados en el esquema del equipo receptor. El contenido y funcionalidad de estos módulos generales se explicará con mayor detalle en el capítulo 6.

Además, la aplicación principal `MS`, hace uso de un módulo adicional que contiene diversas utilidades:

- `I2B.py`: Debido a que la plataforma PyS60 posee ciertas particularidades con respecto a la versión de Python para PC, ha sido necesario desarrollar este módulo adicional para la manipulación y conversión de números enteros (*integer* o *long*) a cadenas binarias de texto.

Por otra parte, la aplicación accede a diversos datos almacenados en el terminal:

- Archivo de datos: El terminal, o en su defecto el emulador software, guardan y acceden a los datos de conexión del equipo receptor a través del fichero `ms_data.txt`. El objetivo de esto es la no volatilidad de dichos datos entre ejecuciones.

- Logotipo: Al iniciar la aplicación móvil, el terminal, o en su defecto el emulador software, muestra durante unos segundos una pantalla de bienvenida basada en la imagen `ms_logo.jpeg`, a modo de detalle estético.

- Fotografías: Aunque no se muestran explícitamente en la figura 4.2, el terminal, o en su defecto el emulador software, acceden a las imágenes JPEG capturadas mediante la cámara integrada del teléfono para su envío posterior. Concretamente, en el caso de trabajar con el emulador software, dichas imágenes están almacenadas previamente en una carpeta de la memoria del mismo.

Como veremos en el capítulo 5, donde se detalla el software desarrollado para el equipo receptor y sus funcionalidades, la estructuración modular de la aplicación implementada en CH difiere ligeramente de la estructura del MS, ya que el equipo receptor cuenta con dos elementos principales diferenciados: la clase `CH`, encargada de instanciar objetos capaces de crear y lanzar la interfaz gráfica de usuario, y la clase `CHservices`, encargada de crear objetos que permitan dar servicio y gestionar las conexiones del sistema, así como de reproducir los flujos multimedia M-JPEG enviados desde el terminal móvil.

La razón de esta decisión de diseño radica en el módulo `threading` de la librería estándar de Python. Una vez importado, el módulo `threading` permite que, cualquier clase que implementemos, herede de la clase estándar `Thread`, lo que confiere a los objetos que instanciemos el carácter de hilo, lo que permite la ejecución concurrente de sus funciones dentro del sistema, una cualidad indispensable del esquema planteado en el presente trabajo.

De este modo, y como veremos en el capítulo 5, el equipo receptor posee dos clases de objetos diferentes. Una de ellas, encaminada a crear la interfaz gráfica de usuario, y la otra dedicada a ofrecer los diferentes servicios de gestión, conexión y reproducción requeridos en el receptor.

Sin embargo, el módulo estándar `threading` no está disponible de forma predeterminada en la plataforma PyS60 para la serie de terminales Symbian S60, lo que obliga a la utilización de otro módulo Python estándar de bajo nivel denominado `thread` para la creación de hilos en la aplicación móvil. Este hecho implica que los objetos creados no puedan heredar de `Thread`, aunque si que es posible lanzar hilos para una función especifica.

Por otro lado, y aunque según la documentación oficial es posible la instalación del módulo `threading` en PyS60, pequeños cambios en la sintaxis utilizada entre el núcleo del intérprete Python para computadora y el utilizado para su instalación en terminales S60, dificultan este proceso.

En este sentido, la aplicación implementada para el terminal móvil, invita a ser desarrollada mediante el esquema planteado, el cual solo posee un archivo principal, `MS`, cuya programación no está orientada a objetos, si no que sigue un estilo de algorítmica secuencial que aprovecha las capacidades ofrecidas por

el módulo estándar de bajo nivel `thread`, suficiente para satisfacer las necesidades requeridas por el sistema. Por estas razones, aunque en un principio se pensó plantear un esquema similar en ambos extremos, al final se optó por la estructura presentada en el presente capítulo.

4.1.1. Funciones *MS*

A continuación se describen brevemente las funciones contenidas en el programa principal `MS`, mostradas en la figura 4.1.

start()

La función `start`() solicita establecer una sesión SIP multimedia para el envío de un flujo M-JPEG y gestiona su actualización en caso de *hand-off* o cambio de red.

sendFrames()

La función `sendFrames`(*rtp_port*) se encarga de enviar un flujo continuado de imágenes M-JPEG sobre RTP hacia el puerto indicado (pasado por parametro).

hand_off()

La función `hand_off`() se encarga de simular un cambio de red o procedimiento *hand-off*.

stop()

La función `stop`() se encarga de finalizar la aplicación `MS`, así como los hilos de ejecución lanzados por el sistema móvil.

exit_handler()

La función `exit_handler`() representa un manejador de eventos asociado a la tecla `Exit` del terminal. Se encarga de controlar en que momento el usuario pulsa dicho botón para salir del sistema.

gui()

La función `gui()` se encarga de crear el menú principal de la interfaz gráfica de usuario de la aplicación móvil, así como sus opciones.

setIP(), setPort, getIP(), getPort()* y *showIpPort()

Este conjunto de funciones se encarga de fijar, mostrar y obtener los parámetros de conexión configurados en la aplicación `MS`.

4.2. Funcionamiento

En la figura 4.3 podemos ver de forma simplificada el funcionamiento de la aplicación implementada en el terminal móvil.

Como ya comentamos en la sección 4.1, el usuario carga la aplicación móvil mediante la ejecución del archivo `MS.py`, cuyo código es ejecutado por el intérprete Python para S60.

Tras una pantalla de bienvenida, el sistema carga el menú y los controles de la aplicación. A cada opción disponible de la interfaz gráfica, se le asigna una funcionalidad determinada mediante la ejecución de la función `gui()`. Inmediatamente después, el sistema alcanza una posición de reposo, en espera de las acciones del usuario.

Llegado este momento, el usuario decide iniciar una sesión multimedia con el equipo receptor predeterminado. Para ello, el sistema invoca a la función `start()` de `MS`, lo que inicia el correspondiente intercambio de mensajes SIP `INVITE`, para dar comienzo posteriormente al envío del flujo de imágenes M-JPEG desde el MS mediante la función `sendFrames()`, la cual se invoca de manera concurrente mediante la instrucción:

```
thread.start_new_thread(sendFrames, (CH_RTP_PORT,))
```

En pleno envío de dicha secuencia, el usuario decide simular un cambio de red o procedimiento *hand-off* pulsando a opción del menú correspondiente, lo que invoca a la función `hand_off()` de la aplicación, la cual provoca que el terminal móvil adquiera una nueva dirección IP de numeración aleatoria y dentro del rango disponible en *localhost*.

Este cambio de red simulado, implica que el MS deba contactar de nuevo con CH desde su nueva ubicación para informarle de dicho cambio, así como renegociar

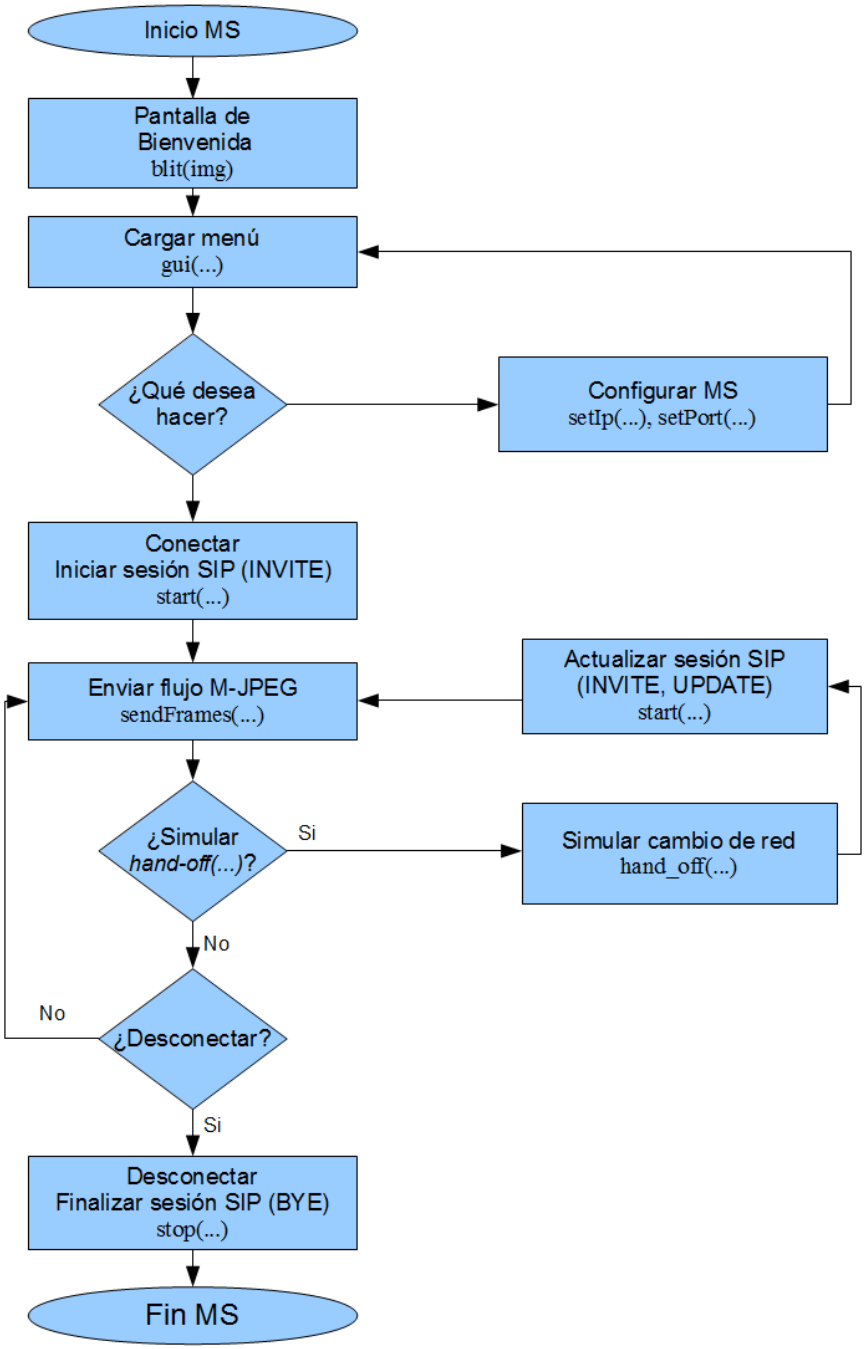

Figura 4.3: Diagrama de flujo de funcionamiento de la aplicación MS.

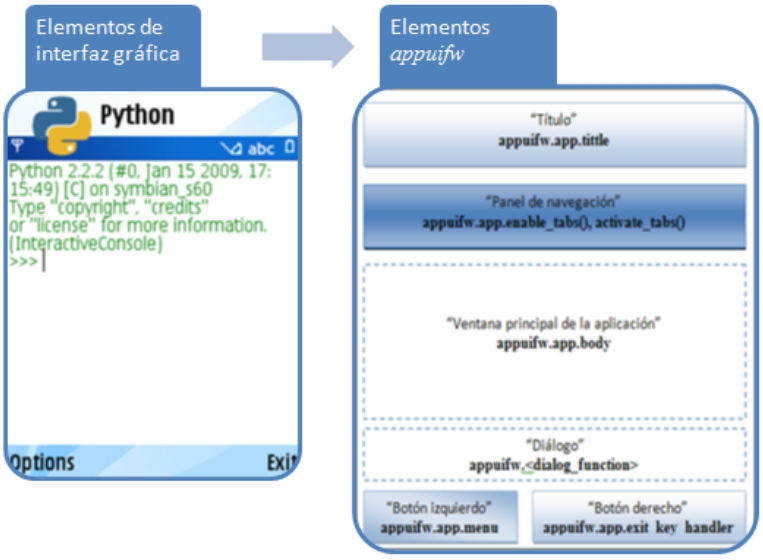

Figura 4.4: Elementos de la interfaz *appuifw*.

la nueva conexión a donde se enviará la secuencia o flujo de imágenes JPEG. Para ello, la función `start()` inicia un procedimiento SIP para el intercambio de mensajes de actualización del tipo `INVITE` y `UPDATE`. Si dicho intercambio de mensajes SIP resulta satisfactorio, el equipo receptor podrá seguir reproduciendo el flujo enviado por el terminal móvil desde su nueva ubicación.

Por último, en pleno envío de dicha secuencia, el usuario móvil decide finalizar la conexión y parar el envío del flujo M-JPEG. En ese momento, el sistema invoca a la función `stop()`, que se encarga de cerrar la sesión de datos con el equipo receptor (mediante el envío de los correspondientes mensajes SIP `BYE`), finalizar el sistema y terminar la ejecución de los hilos que aun queden vivos en la aplicación.

4.3. Interfaz gráfica de usuario

La interfaz gráfica de usuario, de la aplicación del terminal móvil, está implementada mediante las funciones proporcionadas por el módulo PyS60 `appuifw`, el cual nos ofrece la posibilidad de mostrar por la pantalla del dispositivo cuadros de diálogo, listas de selección, notas, mensajes emergentes tipo *pop-up*, etc.

Como se muestra en la figura 4.4, el propio interfaz de Python para S60 posee una estructura común, que es utilizada por la interfaz gráfica de usuario de nuestra aplicación.

De manera esquemática, el intérprete de Python tiene un título, un panel de navegación con diferentes pestañas (el cual, por lo general, no será visible), así como una ventana principal de la aplicación, la cual ocupa el área más grande de la pantalla.

En la parte baja de la aplicación encontramos un cuadro de diálogo, así como el botón del menú de opciones (para ejecutar diversas tareas) y el botón para salir de la aplicación.

Cada uno de los elementos anteriores son simplemente una serie de variables especiales que se encuentran en el interior de un objeto `app`, del módulo `appuifw`. Por esta razón es necesario utilizar el nombre completo de cada uno de los elementos (por ejemplo, `appuifw.app.title`) para referenciarlos. A este conjunto de variables se les puede asignar diferentes valores, al igual que a cualquier otra variable Python, con el objetivo de modificar de forma sencilla la interfaz de nuestra aplicación PyS60.

En nuestro caso, algunos de los elementos o funciones de este módulo PyS60, utilizados en la interfaz gráfica del terminal móvil, han sido los siguientes:

1. Título: `appuifw.app.title` permite agregar el título de la aplicación.
2. Cuerpo: `appuifw.app.body` permite asignar el contenido principal de la interfaz.
3. Menú: `appuifw.app.menu` permite asignar determinadas funciones a la tecla `Options`.
4. Diálogos:
   - Menú emergente o *pop-up*: Creado mediante la función PyS60 `appuifw.popup_menu`(*tupla_con_elementos_del_menú, u"texto_de_selección"*), muestra al usuario cualquier tipo de información que deseemos.
   - Mensaje emergente o *note*: Creado mediante la función PyS60 `appuifw.note`(*u"text", "type"*). En nuestro caso particular, se utiliza para generar el menú de configuración de IP y puerto del terminal móvil.
   - Lista: Creado mediante la función PyS60 `appuifw.Listbox`(*tupla_con_los_elementos_del_menú, función_a_ejecutar*), el cual devuelve un número entero con el índice de la opción que pulsó el usuario, lo que permite a la aplicación actuar conforme la opción elegida. En nuestro caso particular, este tipo de menú es utilizado como ventana principal y cuerpo de la aplicación.

Figura 4.5: Pantalla de bienvenida de la aplicación del terminal móvil.

4.3.1. Pantalla de bienvenida

En la figura 4.5 podemos observar la pantalla de bienvenida, que se muestra al usuario cuando este ejecuta la aplicación PyS60 `MS` desarrollada en este trabajo.

Esta pantalla, que se muestra durante unos segundos mediante la llamada a `e32.ao_sleep`(textittime), se genera mediante la asignación de una imagen al cuerpo de la aplicación, gracia a la función `blit`(*img*), donde *img* debe ser una imagen en formato JPEG.

El código fuente de `MS.py` para conseguir este efecto se muestra en el cuadro 4.1.

```
c = appuifw.Canvas()
appuifw.app.body = c
img=graphics.Image.open("c:\\python\\ms_logo.jpg")
c.blit(img)
e32.ao_sleep(3)
```

Cuadro 4.1: Mostrando la pantalla de bienvenida en la aplicación del terminal móvil

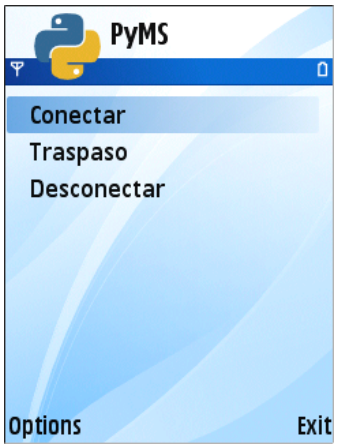

Figura 4.6: Cuerpo de la aplicación del terminal móvil.

4.3.2. Cuerpo de la aplicación

En el caso de nuestra aplicación móvil, asignamos al cuerpo de la interfaz un menú en forma de lista mediante la instrucción:

```
appuifw.app.body =
appuifw.Listbox({[u"Conectar"',u"Cambio de red",u"Desconectar"]},gui)
```

La llamada a `Listbox` asigna la función `gui()` como función de *callback*. A partir de este momento, `gui()` será la función encargada de responder a cada una de las opciones elegidas por el usuario en este menú. De este modo, si el usuario pulsa la opción *Conectar*, la aplicación móvil llamará a la función `start()` de `MS`, si pulsa la opción *Cambio de red*, el sistema invocará a la función `hand_off()`, y si elige *Desconectar* se invoca a la función `stop()`.

En la figura 4.6 podemos observar el resultado final del cuerpo principal de la interfaz de la aplicación en el terminal móvil.

4.3.3. Menú *Options*

Como ya comentamos anteriormente, cada uno de los elementos de una interfaz PyS60 representa una variable determinada dentro del interior del objeto `app`, del módulo `appuifw`.

En el caso de nuestra aplicación móvil, asignamos a la tecla `Options` un menú desplegable mediante la instrucción:

Figura 4.7: Menú *Options*.

```
appuifw.app.menu =
[(u"Direccion", setIp),(u"Puerto", setPort),(u"Ver", showIpPort)]
```

Este menú permitirá configurar la IP y el puerto del sistema receptor en nuestro terminal mediante llamadas a las funciones: `setIP()`, `setPort()`, `getIp()` y `getPort()`, las cuales se encargarán de guardar y cagar dicha configuración desde un fichero de texto denominado `ms_data.txt`.

En la figura 4.7 podemos observar el aspecto final del menú *Options* de la interfaz gráfica de usuario de la aplicación móvil implementada.

4.3.4. Tecla *Exit*

Para que la aplicación móvil PyS60 se mantenga permanente en ejecución, y no finalice hasta que el usuario decida hacerlo, el sistema crea al inicio un cerrojo mediante la instrucción: `lock = e32.Ao_lock()`.

Cuando el usuario inicia la aplicación, este cerrojo se deja a la espera de un evento de salida mediante la instrucción: `lock.wait()`.

A continuación, se asigna un manejador de eventos a la tecla `Exit` del terminal, de tal manera que cuando el usuario pulse dicha tecla del dispositivo, la aplicación pueda capturar dicho evento y ejecutar el manejador asignado.

Para asignar dicho manejador a la telca `Exit` de la interfaz se ejecuta la instrucción:

```
appuifw.app.exit_key_handler = exit_handler
```

La función de *callback* `exit_handler()` se encargará de liberar el cerrojo de la interfaz mediante una llamada a `signal()`, lo que provocará que la aplicación termine, así como de llamar a la función `stop()` para asegurarse de que todos los hilos del sistema finalizan adecuadamente.

Este manejador, tendrá el código fuente mostrado en el cuadro 4.2.

```
def exit_handler():
    thread.start_new_thread(stop,()) # Apagamos todo el sistema
    print "Sistema: Off"
    lock.signal() # Cerramos GUI
```

Cuadro 4.2: Manejador de evento de la tecla *Exit*

4.4. Obtención de dirección IP y registro del terminal

4.4.1. Obtención de dirección IP

Tras la inicialización de las variables del programa, la primera tarea del terminal móvil, o en su defecto el emulador software, es obtener una dirección IP que le permita el acceso a la red para el envío y recepción de información.

En un teléfono real, la tarea de acceder a la red se haría por medio del punto de acceso predeterminado del terminal y de las funciones PyS60 del módulo estándar `socket`: `select_access_point`() y `access_point`(*apid*).

En ese caso, la estación móvil establecería una comunicación con un servidor DHCP preconfigurado para obtener una dirección IP válida que le permita el acceso a la red. Como puede observarse en la figura 4.8, en estos casos, el dispositivo móvil envía un mensaje *broadcast* `DHCP DISCOVER` que le permita descubrir los servidores DHCP de su zona. Los servidores disponibles se comunicarían con el terminal mediante el mensaje DHCP `OFFER`, el cual pone a disposición del dispositivo una nueva dirección IP, la dirección predeterminada de la pasarela o *gateway* que permitirá la conexión a la red, una máscara de red, el nombre de dominio, la dirección local del *proxy* SIP, así como otra clase de parámetros que se estimen convenientes. Llegados a este punto, el terminal móvil devolvería un

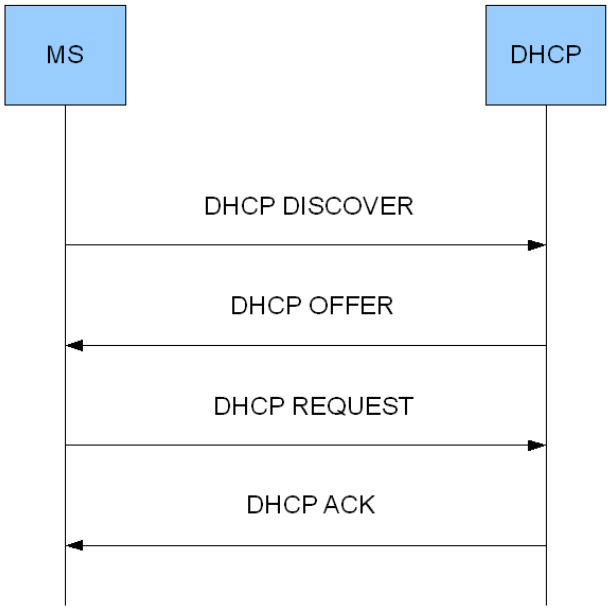

Figura 4.8: Configuración MS mediante DHCP

mensaje DHCP `REQUEST` al servidor elegido, anunciando su selección, y el servidor DHCP devolvería a la estación un mensaje DHCP ACK confirmando la configuración.

Sin embargo, en nuestro caso particular, la IP inicial de nuestro MS será una dirección asignada de forma aleatoria dentro del rango del bucle local de nuestra máquina o *localhost*. Dicha dirección IP inicial será fijada por la aplicación `MS` en la fase de arranque del sistema móvil.

4.4.2. Registro del terminal

En un terminal real, antes de iniciar una sesión SIP multimedia con otro equipo, el MS y el CH deberían registrarse en su *proxy* SIP predeterminado. Existirían tres posibles casos para llevar a cabo este proceso:

1. Registro rápido (ver figura 4.9).
2. Registro completo en la red hogar (ver figura 4.10).
3. Registro completo en la red visitada (ver figura 4.11).

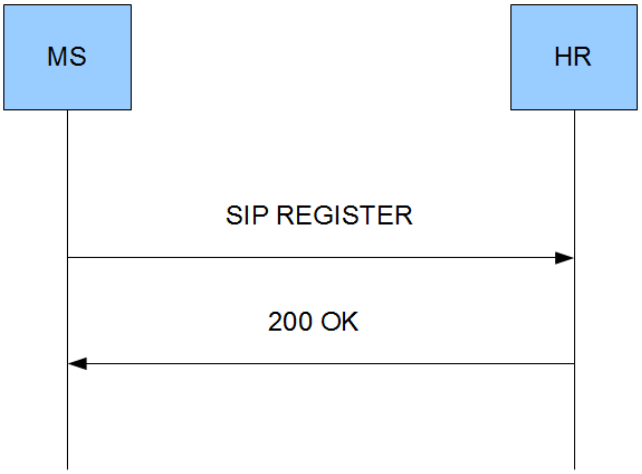

Figura 4.9: Registro rápido.

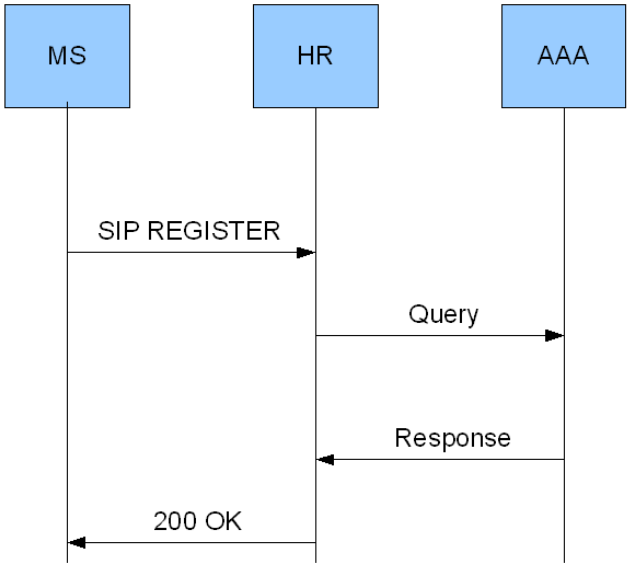

Figura 4.10: Registro completo en la red hogar.

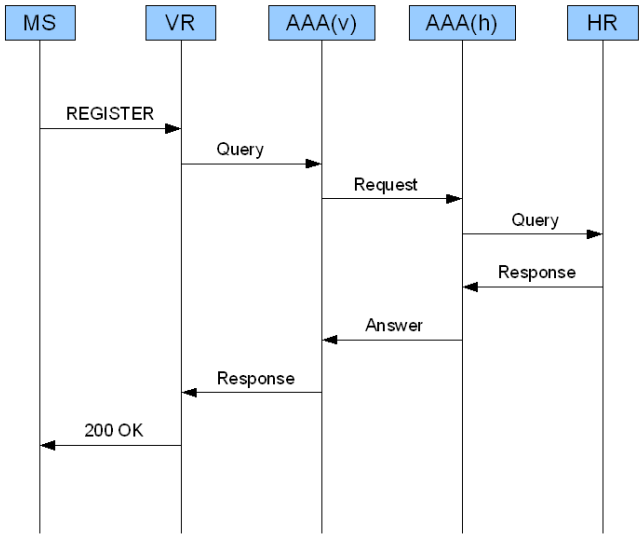

Figura 4.11: Registro completo en la red visitada.

Sin embargo, en nuestro caso particular, y aunque el MS podría dotarse fácilmente de la lógica suficiente para realizar este procedimiento de registro, ya que el módulo general del sistema `SIPmessages` cuenta con dicho tipo de mensajes SIP, este mecanismo queda fuera de los límites de estudio del presente proyecto, ya que este estudio se ciñe exclusivamente a los extremos finales de una conexión para el envío y reproducción de una secuencia M-JPEG, enmarcada en el ámbito de la movilidad en Internet mediante el protocolo SIP.

Por tanto, el procedimiento de registro, el cual involucra a una entidad externa denominada *proxy* SIP, obviado en el presente trabajo, suponiéndose que en el momento de iniciar una sesión SIP, ambos extremos ya se encuentran debidamente registrados y listos para el establecimiento de dicha conexión, proponiéndose la implementación del mecanismo de registro para una futura actualización del diseño actual.

4.5. Establecimiento de sesiones multimedia

Una vez que el terminal obtiene su dirección IP inicial, y con el dispositivo configurado debidamente, el usuario pulsa la opción *Conectar* del menú principal de la aplicación móvil. En ese momento, la función `gui()` del sistema lanzará el hilo de la función `start()` mediante la instrucción: `thread.start_new_thread(start,())`. Esto iniciará el intercambio de mensa-

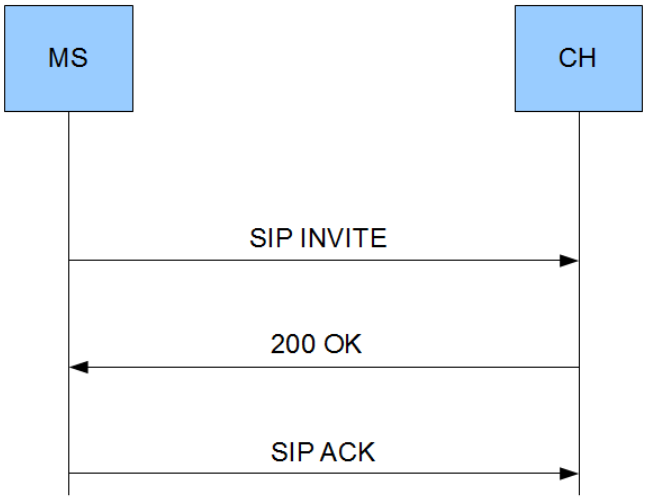

Figura 4.12: Procedimiento SIP para inicio de sesión multimedia.

jes SIP mostrado en la figura 4.12 con el equipo receptor, que previamente debe haber sido configurado y arrancado.

Para ilustrar este procedimiento, vamos a suponer los siguientes parámetros de ejemplo, en el rango de direcciones *localhost*:

- El terminal móvil posee una dirección IP inicial `127.128.189.123`, la cual se indica en el campo "`o`" (originador de la sesión multimedia) del mensaje SDP adjunto en el mensaje SIP `INVITE`. Si el terminal móvil cambia de red, deberá adquirir una nueva dirección IP.
- El equipo receptor posee una dirección IP fija `127.0.0.1` conocida por el terminal móvil, la cual se indica en el campo "`c`" (información de conexión) del mensaje SDP adjunto en el mensaje SIP `INVITE`.
- El terminal móvil indica al equipo receptor que le enviará un flujo de información multimedia con formato M-JPEG al puerto `42449`, el cual se indica en el campo "`m`" (nombre multimedia y dirección de transporte) del mensaje SDP adjunto en el mensaje SIP `INVITE`.

Dicha secuencia de mensajes SIP, intercambiados entre MS y CH, se detalla a continuación:

```
CH <---------- MS
INVITE sip:ms@localhost SIP/2.0
```

```
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 1 INVITE
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Date: Mon, Oct 11 20:31:49 2010 GMT
Content-Type: application/sdp
Content-Length: 157


v=0
o=ms 27246875 458932 IN IP4 127.128.189.123
s=Sesion SDP
i=Flujo M-JPEG sobre RTP
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
m=video 42449/1 RTP/AVP JPEG

CH ----------> MS
SIP/2.0 200 OK
To: <sip:ms@localhost>
From: <sip:ch@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 1 INVITE
Contact: <sip:ch@localhost>; expires 0
User-Agent: Equipo E.P.S. Universidad Carlos III de Madrid
Content-Length: 0

CH <---------- MS
ACK sip:ch@localhost SIP/2.0
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 1 ACK
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
```

```
Date: Mon, Oct 11 20:31:49 2010 GMT
Content-Length: 0
```

La función `start()` utilizará los módulos comunes `SIPmessages` y `SDPmessages`, descritos en el capítulo 6, para la creación de estos mensajes de texto.

Si este procedimiento resulta satisfactorio se establece una sesión SIP entre ambos extremos, y el envío de datos por parte del terminal móvil puede comenzar.

Para ello, la función `start()` lanza un nuevo hilo denominado `sendFrames()` mediante la instrucción `thread.start_new_thread(sendFrames,(rtp_port,))`, que recibe un parámetro *rtp_port* que indica al equipo receptor el puerto donde se emitirá el flujo de imágenes M-JPEG.

Cabe destacar que, como ya comentamos anteriormente en la sección 4.1, la aplicación utiliza el módulo estándar de bajo nivel `thread`, en lugar de `threading`, para lanzar los hilos de servicio, por venir incluido de forma predeterminada en la plataforma PyS60, además de que la instalación de `threading` en el terminal (o en el emulador software) presenta ciertas incompatibilidades en la práctica.

4.6. Envío de flujo de imágenes M-JPEG

El envío de imágenes se produce cuando el terminal ha establecido o actualizado una sesión SIP para la transmisión de información multimedia al equipo receptor.

Llegados a este punto, la función `start()` lanzará un hilo denominado `sendFrames`(*rtp_port*) para servir el flujo M-JPEG en un puerto determinado.

En un terminal real, la función `sendFrames`(*rtp_port*) comenzaría la transmisión de un flujo de imágenes tomadas directamente de la cámara del terminal, y codificadas en en formato JPEG.

Los terminales de la serie S60 poseen diferentes modelos de cámaras, cada una de ellas con diferentes capacidades. Por este motivo, el módulo PyS60 `camera`, ofrece la posibilidad de usar diferentes funciones para obtener información acerca del *hardware* disponible en el teléfono móvil.

En la tabla 4.3 se resumen algunas de las funciones disponibles en este módulo.

Función	Descripción

cameras_available()	Devuelve el número de cámaras disponibles en el dispositivo.
image_modes()	Devuelve una cadena de caracteres con los modos de imagen soportados, como por ejemplo: RGB o JPEG_Exif.
image_sizes()	Devuelve la resolución en una tupla del estilo: (640, 480), (160, 120), etc.
flash_modes()	Devuelve una lista de *strings* con los modos de flash disponibles.
max_zoom()	Devuelve un entero con el máximo zoom digital soportado por el dispositivo.
white_balance_modes()	Devuelve una lista de *strings* con los ajustes de exposición.

take_photo([*mode*, *size*, *zoom*, *flash*, *exposure*, *white_balance*, *position*])	Toma una fotografía y devuelve la imagen. Por ejemplo, para tomar una imagen a resolución 640x480 podemos implementar:
	`def make_photo():`
	
	`image=camera.take_photo(size=(640,480))`
	
	`c=appuifw.Canvas()`
	
	`s=c.size`
	
	`c.blit(image,target=(0,0,s[0],(s[0]/4*3))`
	`,scale=1)`
	
	`# Mediante la linea c.blit(...)`
	`# especificamos el área y la posición`
	`# donde será tomada la nueva foto,`
	`# así como su proporción, 4:3`
	
	`image.save('e:\\Images\\picture.jpg')`

release()	Después de invocar a esta función, la cámara del dispositivo está lista para ser usada por otra aplicación.
start_record(*filename*, *callable*)	Comienza la grabación de video en el dispositivo.
stop_record()	Finaliza la reproducción de vídeo.

Cuadro 4.3: Funciones del módulo camera

En el caso particular de nuestra aplicación, y al haberse realizado las pruebas en el emulador software proporcionado por el SDK de *Nokia Developers Tools*, las imágenes a enviar estarán almacenadas previamente en la memoria del dispositivo, en la ruta `c:\python\`*imagen.jpg*, que equivale a la ruta de sistema `C:\Symbian\9.2\S60_3rd_FP1\Epoc32\winscw\c\python\`*imagen.jpg*. Estas imágenes se enviarán en forma de bucle infinito para su posterior reproducción en el equipo receptor.

4.6.1. Envío de flujo de imágenes sobre RTP y M-JPEG

La función `sendFrames`(*rtp_port*) abrirá una imagen y calculará el número de paquetes necesario para enviarla sobre UDP/RTP, en base a la variable global de la aplicación denominada `RTP_PACKET_SIZE` y el número de octetos de la fotografía. Esto permitirá calcular el *padding* o el número de octetos de relleno necesarios para el último paquete RTP que enviemos de dicha imagen.

Posteriormente, la función `sendFrames`(*rtp_port*) se encargará de cargar cada uno de los fragmentos de la imagen mediante la función `f.read(PACKET_SIZE_RTP)`, donde `PACKET_SIZE_RTP` indica el número de octetos a leer, y los enviará mediante el protocolo RTP y M-JPEG al equipo receptor para su reensamblado y reproducción.

Cabe destacar que la aplicación está diseñada para enviar imágenes a una resolución de 320x240, información que contienen las cabeceras de los paquetes M-JPEG. Y además, el sistema deberá acceder a dichas imágenes en modo "rb" (*read binary* o lectura binaria) para su correcto envío por la red.

Todos los fragmentos enviados contendrán un campo con el número de secuencia RTP y su *offset* M-JPEG o desplazamiento (dentro de una misma imagen), los cuales serán utilizado por el equipo receptor para descartar imágenes que lleguen de forma incorrecta por retardo o por la pérdida de fragmentos en la red. A su vez, todos los datagramas RTP contendrán el número 26 en su campo *Payload Type*, lo que según la RFC 3551 [27] indicará al equipo CH que el flujo transportado está codificado en formato M-JPEG.

El último fragmento enviado de una misma imagen contendrá el bit M de la cabecera RTP a un valor `M=1`, para indicar al receptor que dicho paquete contiene el fragmento final de una imagen completa y que, por tanto, dicha imagen ya estaría lista para su reproducción.

Por otra parte, si el último fragmento enviado de una misma imagen contiene *padding*, el bit P de la cabecera RTP se configurará al valor `P=1`, lo que indicará que el último octeto de datos contiene el número de bytes de relleno incluido en dicho paquete, por lo que el receptor se verá obligado a procesarlo adecuadamente para recuperar los datos originales pertenecientes a la imagen JPEG.

La función `sendFrames`(*rtp_port*) utilizará los módulos comunes `RTPpackets` y `MJPEGpackets`, descritos en el capítulo 6, para la creación de las cabeceras de estos paquetes.

4.6.2. Envío de flujo de imágenes sobre TCP y HTTP

Aunque este procedimiento no ha sido incluido en el desarrollo final de este proyecto, en una etapa temprana de este trabajo se investigó la posibilidad de enviar las secuencias de fotografías capturadas (o almacenadas) utilizando el protocolo HTTP sobre datagramas TCP/IP, siguiendo el estándar establecido para el envío de secuencias de imágenes procedentes de una cámara IP convencional.

A pesar de todo, finalmente se decidió realizar el envío del flujo M-JPEG sobre UDP y RTP descrito en la sección 4.6.1, debido a que la combinación de TCP y HTTP resulta ineficiente en un esquema de red móvil, donde los paquetes pueden sufrir un retardo y *jitter* significativos.

De todas formas, se ha decido reseñar esta línea de trabajo complementaria para ilustrar su funcionamiento.

Para iniciar este tipo de secuencia es preciso el envío de un primer mensaje HTTP, por parte del MS al CH, con el formato mostrado en el cuadro 4.4.

Como puede observarse en el cuadro 4.4, este primer mensaje contiene diversas cabeceras HTTP con información acerca del contenido multimedia transportado, así como la primera imagen codificada en el cuerpo del mismo:

1. `Content-type` del mensaje general: Campo obligatorio, donde deberá indicarse que el tipo de datos multimedia transportado por HTTP forma parte de un mensaje múltiple, así como el tipo MIME. El tipo de contenido `multipart/x-mixed-replace`, desarrollado como parte de una tecnología para emular a un servidor tipo *push streaming* sobre HTTP, dictamina que todas las partes de un mensaje tipo *mixed-replace* poseen el mismo significado semántico. Sin embargo, cada parte "remplaza" a la parte previa tan pronto como es recibida completamente. Por tanto, el receptor debe procesar la parte individual al momento de su llegada y no debe esperar a que

```
HTTP/1.1 200 OK
Content-type: multipart/x-mixed-replace;
boundary=--myboundary

--myboundary
Content-Length: 6280
Content-type: image/jpeg

JPEG DATA STARTS HERE
fdañfsjlkejfñanvdvnjagknhag5a1a1fa
...
```

Cuadro 4.4: Primer mensaje del envío de una secuencia de imágenes JPEG sobre HTTP desde el terminal móvil

termine el mensaje completo.

2. `boundary=`: Contiene la frontera que permitirá al CH diferenciar entre las diferentes imágenes enviadas dentro de la secuencia. Dicha frontera será una cadena de texto que no podrá ser repetida en ninguna otra parte del mensaje.

3. `Content-Length`: Contiene la longitud de los datos enviados.

4. `Content-type` de cada una de las partes del mensaje: Contiene el tipo particular de datos multimedia enviados, en nuestro caso una secuencia de imágenes en formato JPEG, por lo que su valor será `image/jpeg`.

Es importante destacar que en este tipo de esquemas, se hace preciso la creación de una frontera (*boundary*) para separar y delimitar adecuadamente las imágenes enviadas. De esta forma, y a modo de ejemplo, la cadena `--myboundary` no podrá repetirse en ninguna otra parte del cuerpo del mensaje, ya que en caso contrario, induciría a error en la aplicación receptora a la hora de decodificar la secuencia recibida.

Tras el envío de este primer mensaje HTTP, el resto de datagramas de la secuencia de imágenes tendrán el aspecto mostrado en el cuadro 4.5.

Como puede observarse en dicho cuadro 4.5, tras el primer mensaje, tan solo es necesario volver enviar las cabeceras HTTP particulares de cada imagen, in-

```
--myboundary
Content-Length: 6280
Content-type: image/jpeg

JPEG DATA STARTS HERE
fdañfsj1542lkehahafddnjagknhag5a1a1fa
...
```

Cuadro 4.5: Envío de una secuencia de imágenes JPEG sobre HTTP desde el terminal móvil

cluyendo su tamaño en el campo `Content-Length` y el tipo en `Content-type`, así como la propia fotografía.

Para el envío de mensajes HTTP, en PyS60 contamos el módulo Python `httplib`, el cual nos permite subir fotografías desde nuestro terminal a una URL concreta mediante las sentencias. Veamos un ejemplo para ilustrar esta idea en el cuadro 4.6.

```
import appuifw, httplib

#ATENCIÓN: El código de este ejemplo debe verificarse con una URL y un servidor PHP válidos.

def upload_to_url():
    file = open('e:\\Images\\picture.jpg', 'rb')
    chunk = file.read()
    file.close()

    headers = {"Content-type": "application/octect-stream", "Accept": "text/plain"}
    conn = httplib.HTTPConnection("www...com")
    conn.request("POST", "/image_upload.php", chunk, headers)
    conn.close()
    appuifw.note(u"Foto enviada")

upload_to_url()
```

Cuadro 4.6: Envío de una imagen a una URL mediante HTTP

El método `POST` necesita definir una serie de cabeceras HTTP conforme al estándar RFC822 [30]. Dichas cabeceras deberían pasarse mediante el parámetro `headers` a la función en el momento de realizar la conexión con el equipo receptor, lo que planteaba una complejidad adicional inecesaria en dicho sistema. Por esta

razón, se optó por un procedimiento más sencillo que no implicara la utilización de este módulo PyS60, y realizar la conexión HTTP y envío directamente, generando los mensajes y las cabeceras HTML anteriormente explicados de forma directa mediante texto plano. Además, aunque el protocolo HTTP utiliza el puerto 80 para realizar las transferencias, cabe la posibilidad de especificar los parámetros de conexión en el momento de su creación.

Concretamente, el envío y toma sucesiva de fotografías se podría realizar mediante el código Pys60 mostrado en el cuadro 4.7.

```
TCPSock = socket.socket(socket.AF_INET,socket.SOCK_STREAM) # Abrimos el socket UDP
TCPSock.bind((IP, PORT))
TCPSock.listen(10)
socket_c,(host_c, puerto_c)=TCPSock.accept()
flag=0

while True:
        image=camera.take_photo(size=(640,480))
        c=appuifw.Canvas()
        s=c.size
        c.blit(image,target=(0,0,s[0],(s[0]/4*3)),scale=1)

        f = open(u'c:\\python\\imagen.jpg', 'r') # Abrimos el video en forma binaria
        numBytes = os.path.getsize(u'c:\\python\\imagen1.jpg')
        buf=f.read()
        f.close()

        http_header="HTTP/1.1 200 OK\n"+"Content-type: multipart/x-mixed-replace;
        boundary=--myboundary\n"+"\n"+"--myboundary\n"+"Content-Length: "+str(numBytes)+
        "\n"+"Content-type: image/jpeg\n"+"\n"

        packet=http_header+buf
        socket_c.send(packet)

        packet="--myboundary\n"+"Content-Length: "+str(numBytes)+"\n"+
        "Content-type: image/jpeg\n"+"\n"
        packet=packet+buf
        socket_c.send(packet)

        if flag==1:
         break

socket_c.close()
TCPSock.close()
```

Cuadro 4.7: Captura y envío de imágenes JPEG sobre HTTP desde el terminal móvil

Cabe destacar que las imágenes capturadas por el terminal deben abrirse en modo "rb" (*read binary*) para su correcto envío. Tras almacenar la imagen en una variable, se establece una conexión HTTP y se procede al envío desde el MS.

4.7. Simulación de *hand-off* o cambio de red

4.7.1. Detección de *hand-off* o cambio de red

En un terminal real, cuando un usuario cambia de ubicación y el sistema detecta que el nivel de señal recibido es insuficiente mediante la función `signal dbm()` del módulo PyS60 `sysinfo`, la aplicación realiza un cambio de subred intradominio o de red interdominio, con lo que el terminal recibe una nueva dirección IP. Este cambio implica que el sistema tenga que notificar al equipo receptor la nueva ubicación lógica dentro de la red y actuar en consecuencia, realizando un proceso de intercambio de mensajes SIP entre el MS y el CH. Este procedimiento es conocido simplemente como *hand-off* o cambio de red.

Existen dos posibles tipos de cambio de red, y el esquema real de detección y reconocimiento de un *hand-off* se implementa desde el supuesto inicial de que un cambio de subred o de dominio implica un cambio de celda como pre-requisito:

1. *Hand-off* de subred: Se realiza cuando la estación móvil se desplaza de una subred a otra pero ambas pertenecientes al mismo dominio administrativo. La estación móvil interactúa con el servidor DHCP predeterminado para reconfigurarse automáticamente. Este proceso tiene una duración de un *round-trip* o viaje de ida y vuelta, es decir, el retardo de propagación entre los elementos implicados MS-DHCP-MS (ver figura 4.8). La estación móvil vuelve a invitar al equipo correspondiente con su nueva dirección temporal, procedente de la asignación anterior. En estos casos, el mensaje `INVITE` del protocolo SIP debería incluir un campo `Record-Route` que contenga un identificador del servidor *proxy* de la red visitada.

2. *Hand-off* de dominio: Se realiza cuando la estación móvil se desplaza a una nueva red perteneciente a otro dominio administrativo. Exceptuando el hecho de que una estación móvil requiere un completo registro para completar el cambio de una red a otra, el procedimiento de *hand-off* entre distintos dominios administrativos es similar al procedimiento *hand-off* entre subredes descrito en el caso anterior. En estos casos, este procedimiento también es conocido como *roaming*.

De forma general, este procedimiento ya se ilustró en la figura 2.3.

En el caso particular de nuestra aplicación móvil, este cambio de red se implementará mediante una función denominada `hand_off()`, que se encargará de simular el cambio de IP en el terminal. Dicha función será invocada cada vez que el usuario móvil quiera simular un cambio de red pulsando la opción *Cambio de*

red, del menú de opciones de la interfaz gráfica de usuario proporcionada por la aplicación `MS`.

4.7.2. Procedimiento *hand-off* o cambio de red

Una vez establecida una sesión multimedia, y en pleno envío de una secuencia de imágenes M-JPEG, el usuario móvil decide pulsar la opción *Cambio de red* del menú principal de la aplicación `MS`. En ese momento, el sistema llama a la función `hand_off()`, la cual es capaz de generar una nueva dirección IP aleatoria dentro del rango de direcciones *localhost* o de nuestra máquina local.

Este procedimiento alertará al hilo principal `start()`, el cual detecta que se ha producido un *hand-off* o cambio de red, ya que la variable global del sistema `ms_ip` habrá cambiado, lo que implica que el terminal móvil avise al equipo receptor de su nueva ubicación, así como de la nueva dirección y puerto donde se encuentra el nuevo flujo M-JPEG enviado. Para ello se utiliza la secuencia de mensajes SIP mostrada en la figura 4.13, detallada en el *IETF Draft* [23].

Para ilustrar este procedimiento, vamos a suponer los siguientes parámetros de ejemplo, en el rango de direcciones *localhost*:

- El terminal móvil posee una dirección IP inicial `127.128.189.123`. Al producirse el *hand-off* o cambio de red, el terminal móvil adquiere una nueva dirección IP `127.99.57.55`, la cual se indica en el campo "`o`" (originador de la sesión multimedia) del mensaje SDP adjunto en el mensaje SIP `INVITE`.
- El equipo receptor posee una dirección IP fija `127.0.0.1` conocida por el terminal móvil, la cual se indica en el campo "`c`" (información de conexión) del mensaje SDP adjunto en el mensaje SIP `INVITE`.
- El terminal móvil indica al equipo receptor que ha cambiado de red, y ambos intercambian una serie de mensajes SIP de tipo `INVITE-UPDATE`. El terminal móvil le notifica al equipo receptor el nuevo puerto `58175` donde localizar el flujo M-JPEG, el cual se indica en el campo "`m`" (nombre multimedia y dirección de transporte) del mensaje SDP adjunto en el mensaje SIP `INVITE`.

Dicha secuencia de mensajes SIP, intercambiados entre MS y CH, se detalla a continuación:

```
CH <---------- MS
INVITE sip:ms@localhost SIP/2.0
```

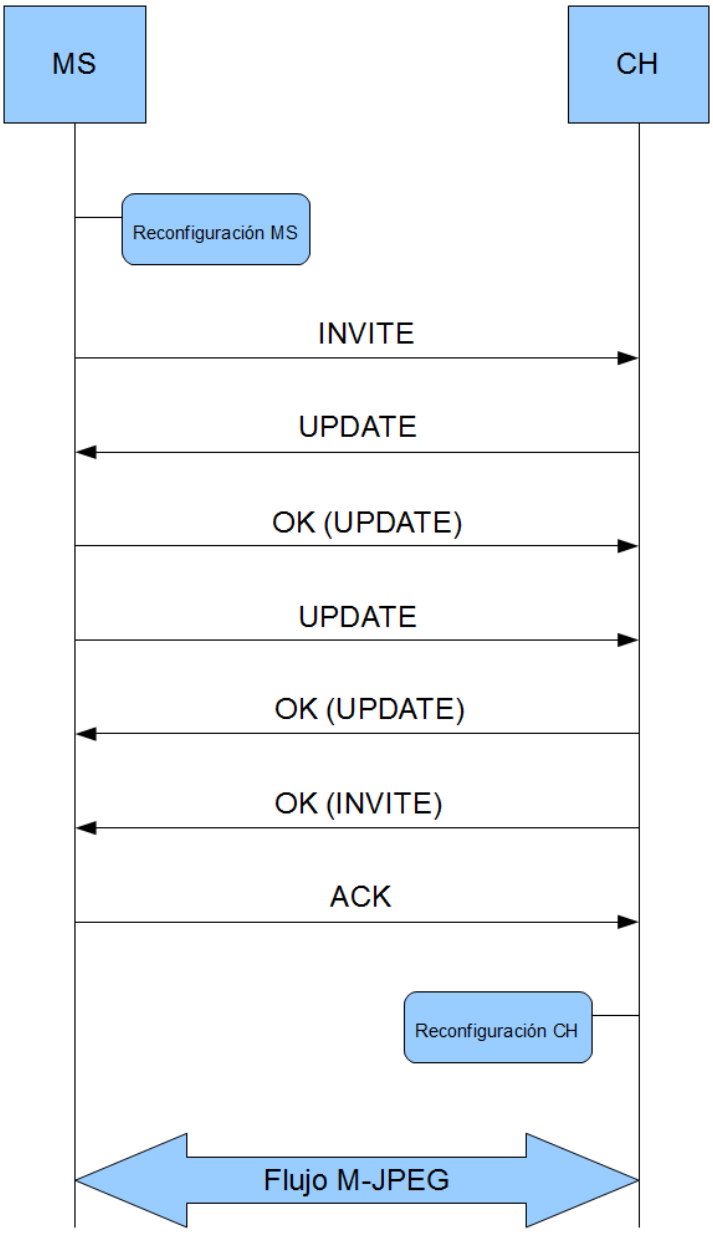

Figura 4.13: Señalización de la aplicación para *hand-off* entre redes.

```
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 INVITE
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Date: Mon, Oct 11 20:32:22 2010 GMT
Content-Type: application/sdp
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
m=video 58175/1 RTP/AVP JPEG

CH ----------> MS
SIP/2.0 UPDATE
To: <sip:ms@localhost>
From: <sip:ch@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 UPDATE
Contact: <sip:ch@localhost>; expires 0
User-Agent: Equipo E.P.S. Universidad Carlos III de Madrid
Date: Mon, Oct 11 20:32:23 2010 GMT
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
```

```
m=video 58175/1 RTP/AVP JPEG

CH <---------- MS
SIP/2.0 200 OK
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 UPDATE
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
m=video 58175/1 RTP/AVP JPEG

CH <---------- MS
SIP/2.0 UPDATE
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 UPDATE
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Date: Mon, Oct 11 20:32:22 2010 GMT
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
e=ms@localhost
```

```
c=IN IP4 127.0.0.1/0
t=0 0
m=video 58175/1 RTP/AVP JPEG

CH ----------> MS
SIP/2.0 200 OK
To: <sip:ms@localhost>
From: <sip:ch@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 UPDATE
Contact: <sip:ch@localhost>; expires 0
User-Agent: Equipo E.P.S. Universidad Carlos III de Madrid
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
m=video 58175/1 RTP/AVP JPEG

CH ----------> MS
SIP/2.0 200 OK
To: <sip:ms@localhost>
From: <sip:ch@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 INVITE
Contact: <sip:ch@localhost>; expires 0
User-Agent: Equipo E.P.S. Universidad Carlos III de Madrid
Content-Length: 163


v=0
o=ms 30578125 458932 IN IP4 127.99.57.55
s=Sesion SDP Hand-off
i=Flujo M-JPEG sobre RTP
```

```
e=ms@localhost
c=IN IP4 127.0.0.1/0
t=0 0
m=video 58175/1 RTP/AVP JPEG

CH <---------- MS
ACK sip:ch@localhost SIP/2.0
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 2 ACK
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Date: Mon, Oct 11 20:32:22 2010 GMT
Content-Length: 0
```

4.8. Finalización de sesiones multimedia

En un momento determinado, y en pleno envío de la secuencia de imágenes M-JPEG, el usuario decide finalizar la sesión multimedia pulsando la opción *Desconectar* del menú principal de la aplicación móvil `MS`. En ese momento, el sistema llama a la función `stop()` de `MS` que avisa a los hilos en ejecución que el programa debe terminar.

La función `start()` iniciará un procedimiento SIP de tipo `BYE`, con el objetivo de informar al equipo receptor de que la sesión ha finalizado y de que se va a interrumpir el flujo M-JPEG enviado desde el terminal.

En la figura 4.14 podemos observar el intercambio de mensajes SIP realizado entre el terminal móvil y el equipo receptor para finalizar la sesión multimedia.

Dicha secuencia de mensajes SIP, intercambiados entre MS y CH, se detalla a continuación:

```
CH <---------- MS
BYE sip:ch@localhost SIP/2.0
To: <sip:ch@localhost>
From: <sip:ms@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
```

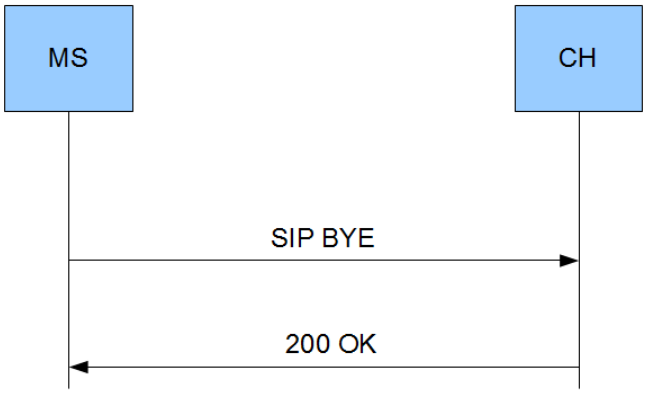

Figura 4.14: Procedimiento SIP para la finalización de una sesión multimedia.

```
CSeq: 3 BYE
Contact: <sip:ms@localhost>; expires 0
User-Agent: Emulador S60 3rd FP1
Date: Mon, Oct 11 20:32:34 2010 GMT
Content-Length: 0


CH ----------> MS
SIP/2.0 200 OK
To: <sip:ms@localhost>
From: <sip:ch@localhost>;tag=0
Via: SIP/2.0/UDP localhost;branch=0;rport
Call-ID: 458932@localhost
CSeq: 3 BYE
Contact: <sip:ch@localhost>; expires 0
User-Agent: Equipo E.P.S. Universidad Carlos III de Madrid
Content-Length: 0
```

Llegados a este punto, la variable global `off` de la aplicación `MS` forzará a que los hilos en ejecución, para el envío de imágenes, finalicen adecuadamente.

Una vez realizado este procedimiento, el usuario podrá decidir si desea salir del sistema pulsando la tecla `Exit` de su terminal, o si por el contrario desea establecer una nueva sesión con el mismo u otro equipo receptor (previa configuración) pulsando de nuevo la opción *Conectar* del menú principal de la aplicación.

Capítulo 5

Arquitectura de la aplicación del equipo receptor

En el presente capítulo, se intentará profundizar en los detalles del software implementado para el equipo receptor mediante el lenguaje de programación Python, el conjunto junto de librerías estándar y la librería externa PIL.

En los siguientes epígrafes se explicarán detalladamente la estructura, interfaces y el funcionamiento de la aplicación propuesta para la parte receptora del sistema. Para ello, a lo largo del presente capítulo se describirán una serie de características técnicas del diseño implementado, las cuales permitirán complementar la visión general sobre el funcionamiento del sistema, ofrecida hasta el momento por el capítulo 3 del presente documento.

5.1. Estructura modular

Durante el presente capítulo centraremos nuestra atención exclusivamente en la parte receptora del sistema implementado (ver figura 5.1) del diagrama completo de clases y módulos (ver figura 3.7 de la sección 3.3.3).

La jerarquía de archivos, que corresponde físicamente al diagrama de la figura 5.1, se ilustra en la imagen 5.2. Ambos esquemas proporcionan una visión completa sobre la estructuración modular de la aplicación desarrollada para el equipo receptor.

La aplicación del equipo receptor cuenta con un fichero principal denominado `CH.py`, que se encarga de iniciar la interfaz gráfica del sistema y permitir al usuario decidir cuando arrancar los servicios proporcionados por CH para la reproducción

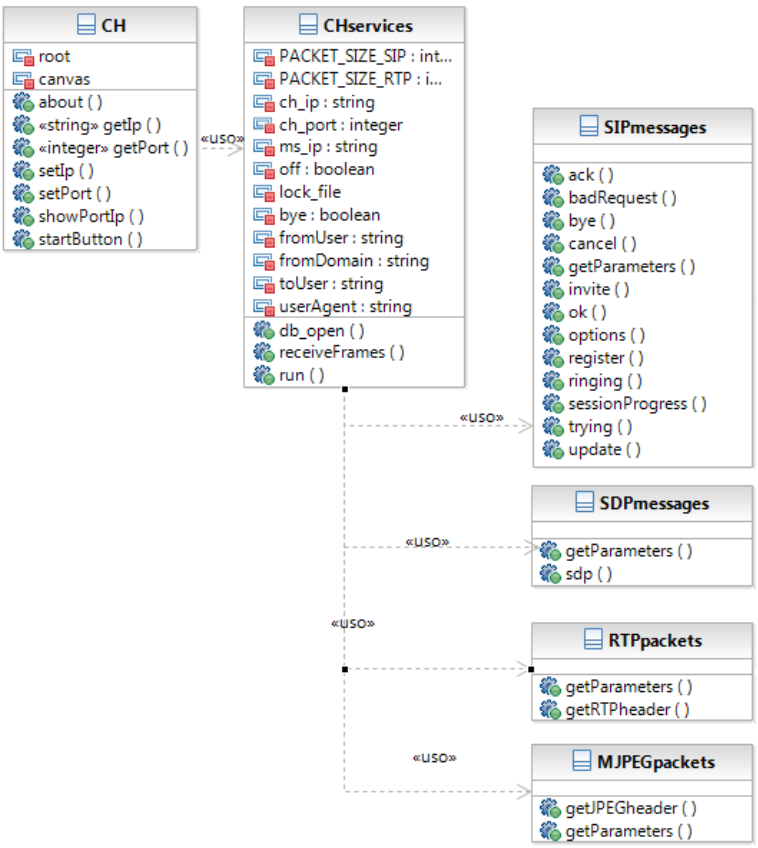

Figura 5.1: Diagrama de clases y módulos de la aplicación receptora.

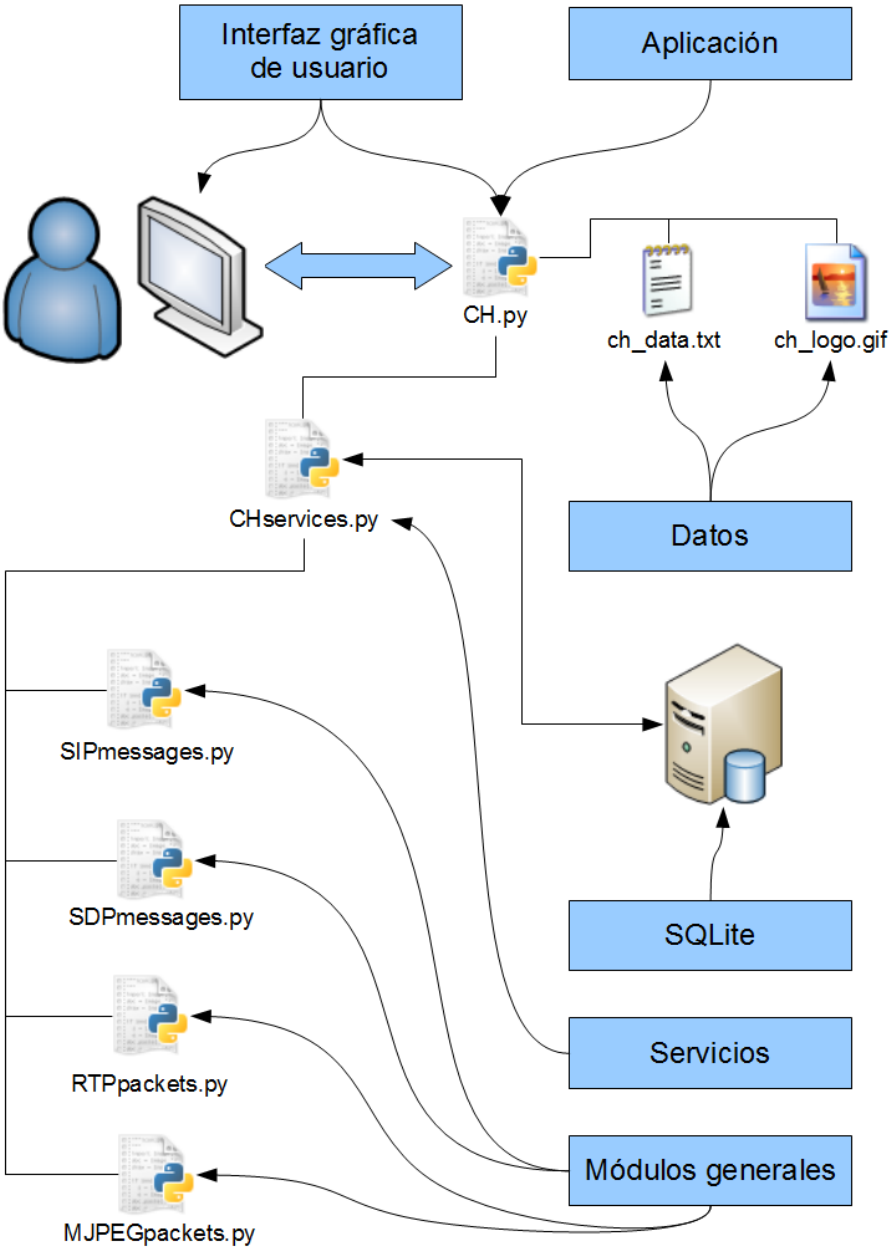

Figura 5.2: Jerarquía de archivos y módulos Python desarrollados para el equipo receptor.

de un flujo multimedia M-JPEG procedente del MS.

Cuando el usuario inicia la aplicación del equipo receptor, el intérprete Python instalado en el equipo se encarga de ejecutar `CH`, lo que inicializará la aplicación, mostrando la interfaz gráfica de usuario que permitirá manejar el software receptor, y de la que hablaremos la sección 5.3.

`CH.py` ofrece la posibilidad de crear objetos la clase `CH` para su posterior manipulación. Concretamente, el sistema crea un solo objeto de esta clase al arrancar, ya que el objetivo principal de esta clase consiste en modelar la interfaz gráfica del sistema receptor, para permitir que el usuario pueda configurar determinados parámetros de conexión (como la dirección IP y el puerto SIP del equipo receptor) y decidir en que momento lanzar el servicio.

Por otro lado, `CH` proporciona un *canvas* o espacio donde poder mostrar las imágenes recibidas, procedentes del MS, a modo de reproductor integrado.

El objeto `CH` importa a su vez la clase `CHservices`, desarrollada en este trabajo, la cual permite instanciar objetos para la manipulación de mensajes SIP/SDP, así como iniciar hilos capaces de servir y mostrar por pantalla las imágenes procedentes de un flujo multimedia M-JPEG (sobre RTP) enviadas desde un terminal móvil.

Por tanto, en el equipo receptor podemos distinguir dos partes diferenciadas: Por un lado `CH`, encargado de generar y controlar la interfaz, y por otro `CHservices`, encargado de prestar distintos servicios a la aplicación (gestionar las conexiones SIP, gestionar los *hand-off* del usuario móvil conectado, reproducir los flujos M-JPEG recibidos, etc.).

Cabe destacar que, la clase `CHservices` hereda a su vez de la clase `Thread`, contenida en el módulo `threading` de la biblioteca estándar de Python, lo que significa que los objetos que instanciemos de dicha clase, se comportarán como un hilo del sistema, permitiendo su ejecución concurrente. En este sentido, debemos mencionar que, por construcción, la aplicación del equipo receptor solo crea un objeto de la clase `CHservices`, lo que implica que durante la ejecución de la aplicación solo existirá un hilo sirviendo las conexiones SIP establecidas en el puerto predeterminado.

Como en el caso de la aplicación desarrollada para el terminal móvil, `CHservices.py` utiliza las funciones proporcionadas por una serie de módulos comunes del sistema, implementados en este trabajo. Por esta razón `CHservices` importa los siguientes módulos Python:

- `SIPmessages.py`.

- `SDPmessages.py`.
- `RTPpackets.py`.
- `MJPEGpackets.py`.

Los módulos comunes del sistema permiten establecer comunicaciones SIP, y recibir flujos M-JPEG sobre RTP procedentes del MS, en el CH, y están pensados para ser reutilizados fácilmente por cualquier otro programa que desarrollemos, siendo reaprovechados en la aplicación de terminal móvil. El contenido y funcionalidad de estos módulos generales ya se introdujo en el capítulo 4, pero se explicará con mayor detalle en el capítulo 6.

Por otra parte, la aplicación accede a diversos datos almacenados en el equipo:

- Archivo de datos: El equipo receptor guarda y accede a los datos de conexión del equipo CH a través del fichero `ch_data.txt`. El objetivo de esto es la no volatilidad de dichos datos entre ejecuciones.
- Logotipo: La aplicación utiliza `ch_logo.gif` a modo de detalle estético, apareciendo empotrado en la interfaz del sistema reproductor.

5.1.1. Funciones CH

A continuación se describen brevemente las funciones contenidas en `CH`, mostradas en la figura 5.1.

__init__()

La función `__init__`() es el constructor de la clase `CH`, que se encarga de invocar al constructor de `Tk` para crear la interfaz gráfica de usuario del equipo receptor.

about()

La función `about`() se encarga de mostrar un mensaje con el *copyright* y diversa información sobre la aplicación receptora.

startButton()

La función `startButton`() representa un manejador de eventos asociado a la tecla `Iniciar` de la interfaz gráfica de usuario del equipo receptor. Se encarga de controlar cuando el usuario pulsa dicho botón, momento en el que `CH` crea un hilo de la clase `CHservices` para comenzar a servir conexiones SIP/SDP.

setIP(), setPort, getIP(), getPort()* y *showIpPort()

Este conjunto de funciones se encarga de fijar, mostrar y obtener los parámetros de conexión configurados en la aplicación `CH`.

5.1.2. Funciones CHservices

A continuación se describen brevemente las funciones contenidas en `CHservices`, mostradas en la figura 5.1.

__init__()

La función `__init__`() es el constructor de la clase `CHservices`, y se encarga de inicializar los servicios necesarios para el establecimiento y actualización de sesiones multimedia con el terminal móvil.

run()

La función `run`() inicializa los servicios SIP/SDP y representa la rutina principal del sistema para el establecimiento de sesiones SIP multimedia, así como para la recepción y reproducción de los flujos M-JPEG enviados desde el terminal.

Además, `run`() es capaz de gestionar la actualización de las sesiones activas mediante la comunicación con una base de datos SQLite, en el caso de que se produzca un procedimiento de *hand-off* en el dispositivo móvil.

db_open()

Por el nivel de seguridad fijado en Python para el trabajo en entornos concurrentes, el acceso a las bases de datos SQite debe tener una conexión independiente por cada hilo. Dicha conexión es establecida mediante la función `db_open`().

receiveFrames()

La función `receiveFrames`(*rtp_port, root, canvas*) se encarga de recibir y reproducir (en el `canvas` del objeto `root`) un flujo multimedia M-JPEG sobre RTP para una dirección IP y un puerto determinado.

5.2. Funcionamiento

En la figura 5.3 podemos ver de forma simplificada el funcionamiento de la aplicación implementada en el equipo receptor.

Como ya comentamos en la sección 5.1, el usuario carga la aplicación reproductora mediante la ejecución del archivo `CH.py`, cuyo código es ejecutado por el intérprete Python para entornos Microsoft Windows.

Al hacer doble *click* sobre el archivo `CH.py`, el sistema carga el menú y los controles de la aplicación, e instancia un objeto de la clase `CH` para su manipulación.

A cada opción disponible de la interfaz gráfica se le asigna un elemento y una funcionalidad determinada, mediante la ejecución de las funciones `pack`() y `add_command`(*label, command*), ambas proporcionadas por las librerías gráficas `Tk`.

Cuando la interfaz gráfica ha cargado completamente, el usuario decide iniciar la aplicación receptora pulsando la tecla *Iniciar* del menú principal, lo que invoca a la función `startButton()` de `CH`. Dicha función instancia un objeto de la clase `CHservices`, que a su vez hereda de la clase `Thread` a través del módulo `threading` de la librería estándar de Python. Esta característica, confiere al objeto creado un comportamiento de hilo, lo que permite su ejecución concurrente en el sistema.

El objeto instanciado de la clase `CHservices` inicia las instrucciones programadas en su función `run()` mediante la invocación a `start()`, poniendo a la escucha un puerto SIP configurado previamente para prestar los servicios requeridos por el sistema, y lanzando una ligera base de datos SQLite para el registro de los usuarios y el control de los flujos M-JPEG activos mediante la función `db_open()`.

Cuando un terminal móvil inicia el correspondiente intercambio de mensajes SIP `INVITE`, previo al envío de un flujo de imágenes M-JPEG, el equipo receptor intercambia las respuestas SIP necesarias para el establecimiento de una sesión multimedia entre CH y MS. Si dicho procedimiento resulta satisfactorio, CH re-

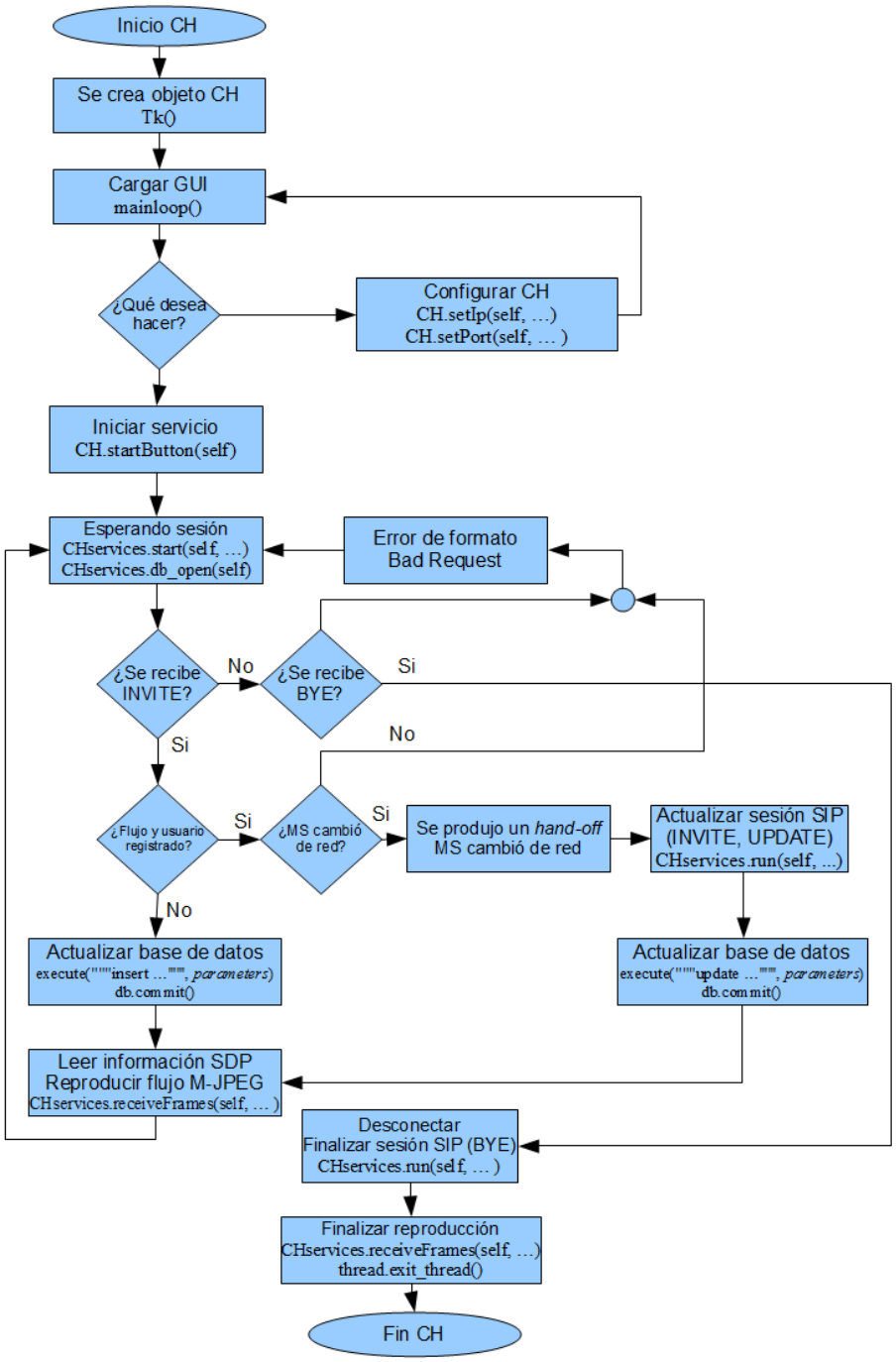

Figura 5.3: Diagrama de flujo de funcionamiento de la aplicación CH.

gistra en el sistema SQLite la ubicación del flujo M-JPEG enviado por el terminal (dirección IP y puerto), así como otros datos de interés como el nombre del usuario emisor o el tipo de datos multimedia enviados por el dispositivo.

Una vez conocidos la dirección IP y el puerto del flujo M-JPEG, a través de la información SDP proporcionada por el terminal móvil en su mensaje SIP `INVITE`, la función `run()` del objeto `CHservices` lanza un hilo de la función `receiveFrames()` mediante la instrucción:

```
thread.start_new_thread(receiveFrames,(rtp_port,root,canvas))
```

Este hilo será el encargado de reensamblar el flujo M-JPEG de un determinado puerto (`rtp_port`) recibido sobre el protocolo RTP, así como mostrarlo en el reproductor integrado (referenciado por el `canvas`) de la interfaz gráfica de usuario (referenciada por `root`).

En pleno envío de dicha secuencia, el usuario móvil decide simular un cambio de red o procedimiento *hand-off* pulsando a opción del menú de su teléfono correspondiente, lo que inicia un proceso de intercambio de mensajes SIP del tipo `INVITE` y `UPDATE`, lo que provoca que el terminal móvil informe al equipo receptor de que ha cambiado de ubicación y ha adquirido una nueva dirección IP dentro del rango disponible en *localhost*.

Este cambio de red simulado, implica que MS y CH deben renegociar la nueva conexión a donde se enviará la secuencia de imágenes M-JPEG. Si dicho intercambio de mensajes SIP resulta satisfactorio, el equipo receptor podrá seguir reproduciendo el flujo enviado por el terminal móvil desde su nueva ubicación, gracias a la información incluida por el dispositivo móvil en las cabeceras de un mensaje SDP (adjunto a un nuevo mensaje SIP `INVITE`).

Por último, en pleno envío de dicha secuencia, el usuario móvil decide finalizar la conexión y parar el envío del flujo M-JPEG. En ese momento, la función `run()` del objeto `CHservices` se encarga de cerrar la sesión de datos con el equipo emisor, mediante el envío de los correspondientes mensajes SIP `BYE`, así como finalizar el sistema y terminar la ejecución de los hilos que aun queden vivos en la aplicación.

5.3. Interfaz gráfica de usuario

La interfaz gráfica de usuario del equipo receptor estará compuesta de dos elementos: El reproductor M-JPEG integrado (ver imagen 5.4), y la salida del intérprete de Python (ver imagen 5.5), la cual utilizaremos para mostrar los mensajes enviados y recibidos por el sistema receptor.

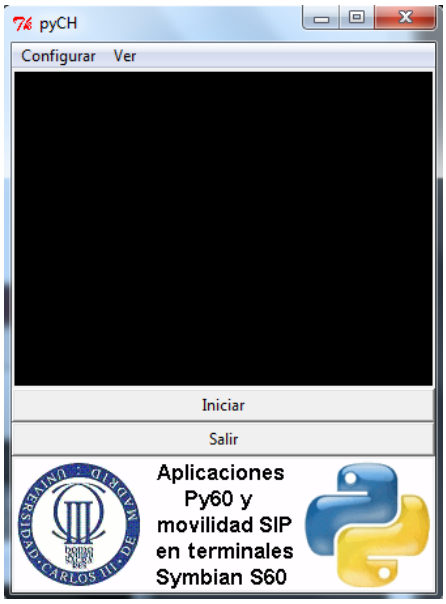

Figura 5.4: Interfaz gráfica de usuario del equipo receptor.

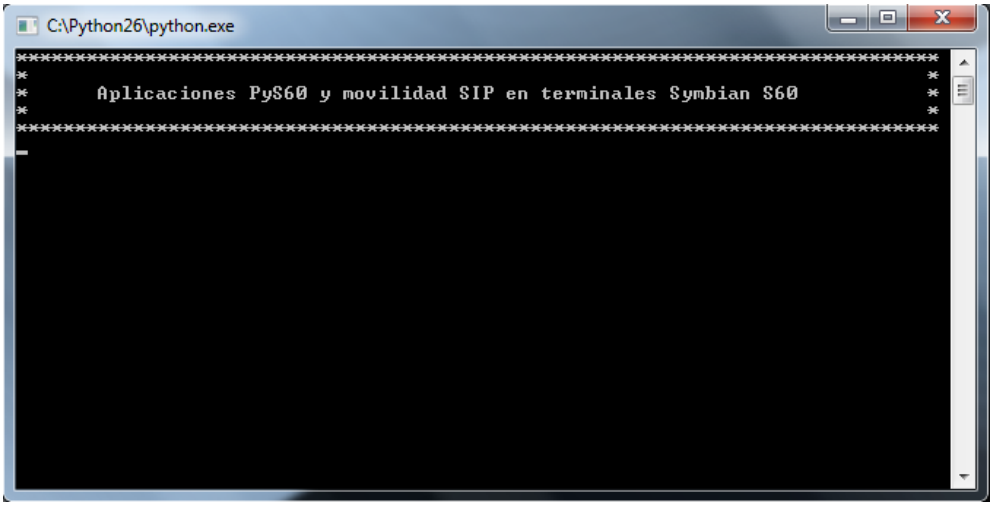

Figura 5.5: Interpréte Python en la aplicación del equipo receptor.

5.3.1. Cuerpo de la aplicación

La interfaz del equipo CH está basada en las librerías estándar `TkInter`. Cuando se lanza la aplicación `CH.py`, el constructor de la clase crea un objeto de la clase `Tk` mediante la instrucción `root = Tk()`.

Gracias la manipulación de dicho objeto, `CH` agrega diversos elementos mediante diferentes funciones para crear la interfaz gráfica del equipo receptor:

- `Frame`: Marco donde se agregarán los elementos de la interfaz gráfica de usuario.
- `Canvas`: Lienzo o espacio para la visualización de las imágenes JPEG recibidas.
- `Menu`: El objeto `Menu` albergará un menú con las opciones de configuración que deseemos.
- `Button`: Mediante los objetos `Button`, instanciados mediante la sentencia `Button`(*toolbar, text, width, command*), podremos agregar distintas funcionalidades (*command*) a las teclas de la interfaz gráfica de usuario.

El cuerpo de la aplicación lo constituye un objeto de la clase `Frame`, el cual contiene un objeto `Canvas` donde se mostrarán las imágenes procedentes de un flujo M-JPEG enviado desde el terminal móvil.

5.3.2. Menú

El ménú, constituido por un objeto de la clase `Menu`, consta de dos partes diferenciadas:

1. `Configurar`: Permite fijar la configuración del equipo receptor, invocando las funciones `setIP()`, `setPort()`, `getIp()` y `getPort()` de la clase `CH`. La información introducida por el usuario del equipo reproductor se recoge mediante la función `TkInter tkSimpleDialog.askstring`(*texto*) y se almacena en `ch_data.txt`.
2. `Ver`: Permite visualizar la configuración del equipo receptor invocando la función `showPortIp()` de la clase `CH`. Además, muestra la licencia de la aplicación mediante la función `about()` de la clase `CH`. La información es mostrada por pantalla mediante la función `TkInter tkMessageBox.showinfo`(*texto*).

5.3.3. Botones *Iniciar* y *Salir*

Los botones de la aplicación han sido creados a partir de objetos de la clase `Button` y representan los controles principales de la aplicación del equipo receptor.

La tecla *Iniciar* permite al usuario decidir el instante en que el servicio debe activarse. *Iniciar* invoca a la función `startButton()` de `CH`, la cual se encarga de instanciar un hilo de la clase `CHservices` e iniciar su ejecución invocando la función `start()` de la clase `Thread`.

Cuando la sesión multimedia ha finalizado, el botón *Salir* invoca a la función del sistema `exit`, la cual se encarga de cerrar la interfaz gráfica y el intérprete Python sobre el que se ejecuta.

5.3.4. Logotipo

Consiste en una imagen GIF, a modo de detalle decorativo de la interfaz gráfica de usuario, la cual se empaqueta en la parte inferior mediante la secuencia de instrucciones mostrada en el cuadro 5.1.

```
photo = PhotoImage(file = "ch_logo.gif")
label = Label(image=photo)
label.pack()
```

Cuadro 5.1: Mostrando el logotipo de bienvenida en la aplicación del equipo receptor

5.4. Establecimiento y actualización de sesiones multimedia

Una que la aplicación receptora se ha iniciado, y la interfaz gráfica se ha cargado correctamente, el usuario activará los servicios del sistema tras pulsar el botón *Iniciar*, el cual invocará a la función `startButton()` de `CH`. Dicha función instancia un objeto de la clase `CHservices`, que a su vez hereda de la clase `Thread` a través del módulo `threading` de la librería estándar de Python. Esta característica, confiere al objeto creado un comportamiento de hilo, lo que permite su ejecución concurrente en el sistema.

El objeto instanciado de la clase `CHservices` inicia las instrucciones programadas en su función `run()` mediante la invocación a `start()`, poniendo a la escucha un puerto SIP configurado previamente (a la espera de nuevas sesiones de datos), y lanzando una ligera base de datos SQLite para el registro de los usuarios móviles y sus flujos M-JPEG activos mediante la función `db_open()`.

Cuando el equipo receptor recibe un mensaje SIP `INVITE` de un usuario móvil, confirma si el flujo M-JPEG procedente de dicho terminal ya había sido registrado en el sistema con anterioridad. Si no es así, el usuario es nuevo y debe registrarse como nueva fuente de imágenes, lo que induce al intercambio de mensajes SIP mostrado en la figura 4.12 (el cual está explicado con mayor detalle en la sección 4.5).

De no ser así, se comprueba si el usuario ha cambiado de red, en cuyo caso se establece el intercambio de mensajes SIP necesario para llevar a cabo el procedimiento *hand-off* requerido. Este cambio de red simulado, implica que el CH debe adquirir la nueva dirección y puerto donde se enviará el flujo M-JPEG. Si dicho intercambio de mensajes SIP resulta satisfactorio, el equipo receptor podrá seguir reproduciendo la secuencia de imágenes enviada por el terminal móvil desde su nueva ubicación, gracias a la información SDP adjuntada por el dispositivo en un nuevo mensaje SIP `INVITE`. Dicho intercambio ya se explicó con detalle en la sección 4.7.2, y puede observarse en la figura 4.13.

Si pasado un tiempo, el terminal móvil cambia red de nuevo, ambos equipos se verán obligados a repetir el anterior procedimiento para actualizar su estado e información de conexión mediante los protocolos SIP y SDP.

5.5. Reproducción de flujo de imágenes M-JPEG

5.5.1. Reproducción de flujo de imágenes sobre RTP y M-JPEG

Mientras `CH` será el objeto encargado de generar la interfaz gráfica del sistema y sus elementos, los hilos `receiveFrames()` lanzados por el objeto `CHservices` serán los encargados de reproducir el flujo M-JPEG recibido. Para ello, se utilizará la librería externa `PIL` (ver anexo B.1.2), la cual es capaz de proporcionarnos el soporte requerido para la manipulación, reproducción y guardado de imágenes JPEG.

Una vez conocidos la dirección IP y el puerto del flujo M-JPEG, a través

de la información SDP proporcionada por el terminal móvil en su mensaje SIP `INVITE`, la función `run()` del objeto `CHservices` lanza un hilo de la función `receiveFrames()` mediante la instrucción:

```
thread.start_new_thread(receiveFrames,(port,root,canvas))
```

Este hilo será el encargado de reensamblar el flujo M-JPEG de un determinado puerto (`port`) recibido sobre el protocolo RTP, así como mostrarlo en el reproductor integrado (referenciado por `canvas`) de la interfaz gráfica de usuario (referenciada por `root`).

Los hilos de servicio `receiveFrames(...)` iniciados por el objeto `CHservices` se encargarán de comunicarse con el objeto `Canvas` de la interfaz gráfica de `CH`, una vez hayan procesado y reensamblado una imagen procedente de un flujo M-JPEG enviado por el termina móvil sobre el protocolo RTP. Tras esto, el hilo en cuestión, invocará el conjunto de sentencias mostrado en el cuadro 5.2 para que la aplicación receptora muestre la imagen por pantalla.

```
img=Image.open("imagen_recibida.jpg", mode='r')
photoimage = ImageTk.PhotoImage(img)
canvas.create_image(320,240, image=photoimage, anchor=SE)
root.update()
```

Cuadro 5.2: Mostrando imagen procedente de un flujo M-JPEG en la aplicación del equipo receptor

El resultado de dicha acción será similar al mostrado en la figura 5.6.

Como puede apreciarse, la aplicación está diseñada para mostrar imágenes a una resolución de 320x240, información que contienen las cabeceras de los paquetes M-JPEG.

5.5.2. Reproducción de flujo de imágenes sobre TCP y HTTP

Aunque este procedimiento no ha sido incluido en el desarrollo final de este proyecto, en una etapa temprana de este trabajo, se investigó la posibilidad de enviar las secuencias de fotografías capturadas o almacenadas utilizando el protocolo HTTP sobre datagramas TCP/IP, siguiendo el estándar establecido para el envío de secuencias de imágenes procedentes de una cámara IP convencional.

Figura 5.6: Aplicación del equipo receptor reproduciendo flujo M-JPEG.

La decisión de diseño, que hizo desechar este esquema definitivamente, se basa en los protocolos utilizados. TCP y HTTP no resultan una buena combinación en un servicio para la reproducción de imágenes en directo, una situación que se agrava por el hecho de enmarcarse en un contexto de redes móviles, donde las pérdidas o las variaciones en el retardo pueden resultar significativamente mayores que un ámbito de red fija.

Por su parte, UDP y RTP hacen que las entregas de paquetes de datos, desde el dispositivo que envía la información a quien la reproduce, se hagan con una velocidad mucho mayor que la que se obtiene por TCP y HTTP. Esta eficiencia es alcanzada por una modalidad que favorece el flujo continuo de dichos paquetes mediante un mecanismo no orientado a conexión.

Cuando TCP y HTTP sufren un error de transmisión, siguen intentando transmitir los paquetes de datos perdidos, hasta conseguir una confirmación de que la información llegó en su totalidad. Sin embargo, un servicio no orientado a conexión como UDP, continúa mandando los datos de forma ininterrumpida, ya que en una aplicación multimedia de estas características, estas pérdidas son relativamente asumibles, descartándose los paquetes necesarios para continuar con la reproducción.

En cualquier caso, ya que en el inicio de este proyecto se investigó este esquema basado en HTTP y TCP, en los siguientes epígrafes detallaremos algunas características técnicas de este mecanismo alternativo.

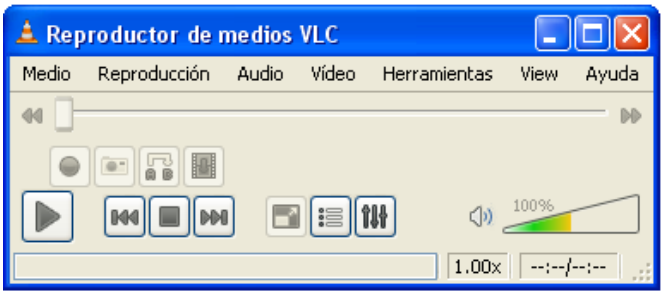

Figura 5.7: Reproductor VLC.

Reproductor VLC

Durante la realización de estas pruebas se decidió utilizar el reproductor multimedia VLC (*VideoLAN Client*), perteneciente al proyecto VideoLAN, a la hora de visualizar el flujo de imágenes M-JPEG enviado sobre HTTP desde el terminal móvil al equipo receptor. Concretamente, se utilizó la versión 1.x de esta aplicación, la cual podemos ver en la figura 5.7.

VLC posee algunas características interesantes, de las que podemos destacar las siguientes:

- VLC es software libre distribuido bajo la licencia GPL. Soporta multitud de *codecs* de audio y vídeo, así como diferentes tipos de archivos, además de DVD, VCD y varios protocolos *streaming*. Utiliza la biblioteca *codec* `libavcodec` del proyecto `FFmpeg` para manejar los muchos formatos que soporta, y emplea la biblioteca de descifrado DVD `libdvdcss` para poder reproducir los DVD cifrados. Además VLC tiene soporte para `Video4Linux`, con lo que representa una de las soluciones gratuitas disponibles actualmente de mayor calidad para realizar la tarea que nos compete en este proyecto.

- Permite su control remoto mediante el protocolo Telnet, lo que facilita de manera significativa la gestión y reproducción de flujos multimedia provenientes de la red mediante simples comandos de texto.

- Python ofrece a los desarrolladores la posibilidad de ejecutar procesos del sistema mediante el módulo `subprocess`, lo que permite controlar la ejecución de los distintos programas instalados en la computadora, facilitando el manejo de las aplicaciones del sistema.

- VLC es uno de los reproductores más independientes, en cuanto a plataforma se refiere, con versiones para GNU/Linux, Microsoft Windows, Mac OS X, BeOS, BSD, Pocket PC, Solaris.

- También puede ser utilizado como servidor *unicast* o *multicast*, en IPv4 o IPv6, en una red de banda ancha, lo que facilita la creación y mejora de futuras implementaciones para nuevos servicios.

Control de VLC mediante el protocolo Telnet

En la figura 5.8 se muestra el intercambio de mensajes entre la aplicación y el reproductor multimedia VLC para establecer el inicio de la reproducción de una secuencia de imágenes enviadas desde el terminal móvil.

El código Python implementado para controlar el reproductor multimedia VLC puede descomponerse en varias acciones diferenciadas, las cuales pasaremos a describir en los siguientes subapartados de la presente sección.

Inicialización de VLC y conexión Telnet

Para lanzar VLC tenemos que invocar a un subproceso del sistema mediante la función `Popen` del módulo Python `subprocess`. Deberán pasarse varios parámetros a dicha función:

1. Ubicación en disco del ejecutable (`.exe`) del reproductor multimedia VLC.

2. Parámetro `'--extraintf'`, que en entornos Microsoft Windows XP indica que se interactuará con VLC mediante una interfaz extra, en nuestro caso el protocolo Telnet.

3. Parámetro `'telnet'`, que indicará al reproductor que se desea controlar mediante el protocolo de comunicaciones Telent.

Posteriormente, la aplicación establecerá una comunicación vía Telent con el reproductor multimedia, utilizando para ello el módulo Python `Telent`, el cual nos permite gestionar de forma sencilla y eficaz este tipo de conexiones.

Una vez establecida la comunicación Telnet entre ambas entidades, la aplicación enviará la contraseña de administración de VLC (por defecto `admin`), la cual garantizará el acceso al control remoto del mismo.

Podemos contemplar un ejemplo en código Python de este procedimiento en el cuadro 5.3.

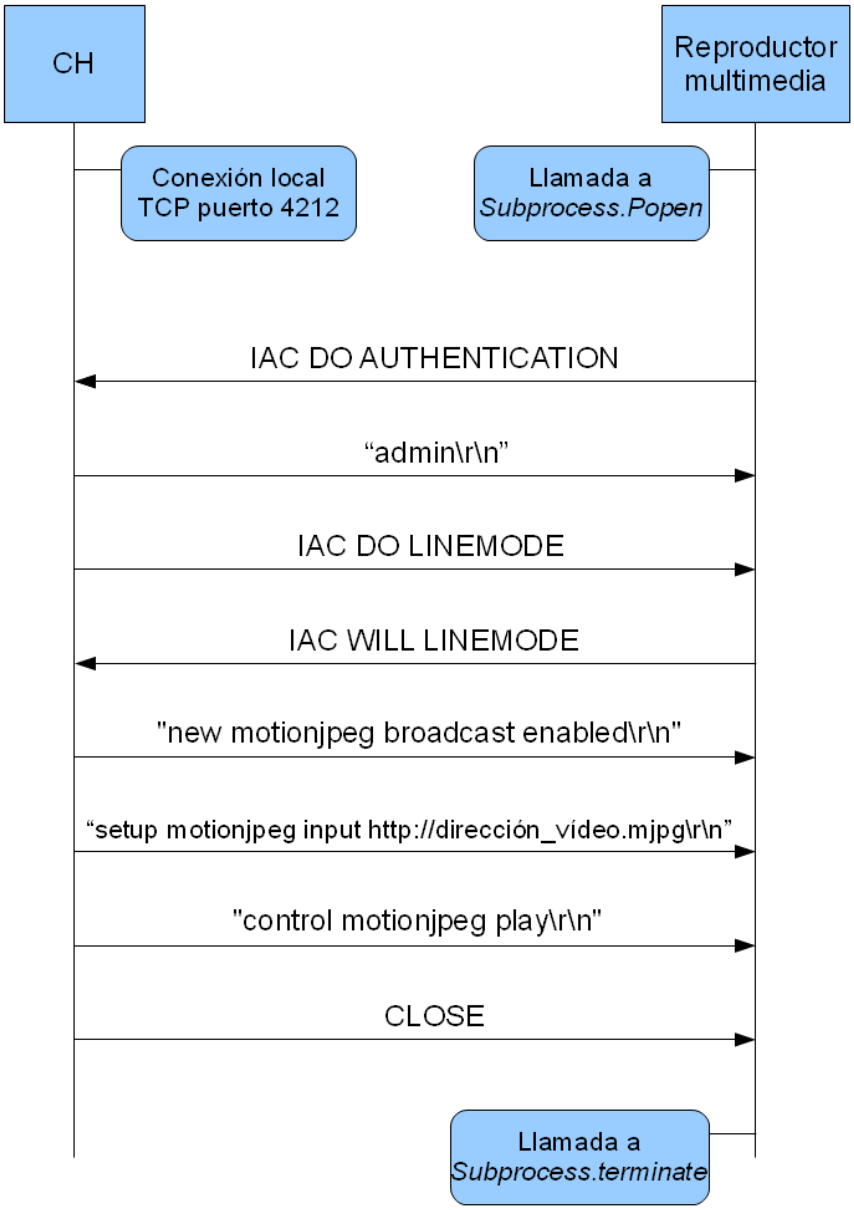

Figura 5.8: Intercambio de mensajes Telnet entre la aplicación y el reproductor multimedia VLC para iniciar la reproducción.

```
import subprocess

...

# Abrir VLC en Windows XP en modo Telnet
vlc=subprocess.Popen(['C:\\Archiv~1\\VideoLAN\\VLC\\vlc.exe', '--extraintf', 'telnet'])

...

# Si todo el proceso ha sido correcto, iniciamos la reproducción del vídeo
# Usamos el puerto por defecto de VLC para el protocolo Telnet (4212)
t=telnetlib.Telnet(host="127.0.0.1", port=4212)

# Introducimos la contraseña de VLC
t.write("admin\r\n")
```

Cuadro 5.3: Inicialización de VLC y control remoto mediante Telnet

Reproducción de una secuencia de imágenes multimedia

Para entender como VLC carga un flujo de datos multimedia, resulta imprescindible conocer los pasos que la aplicación sigue a la hora de cargar el flujo para reproducirlo por pantalla:

1. El primer paso consiste en crear una conexión, indicando a VLC que el tipo de datos esperados en esta misma es de tipo M-JPEG.
2. Posteriormente se indicará una ruta de entrada para dicha conexión. El nombre del flujo es indiferente, no siendo el caso de la dirección IP de la que se recibirá la secuencia de imágenes.
3. Por último, tras enviar la ruta a VLC, cargaremos el flujo multimedia enviando a VLC el comando `play`, el cual iniciará la reproducción.

Podemos contemplar un ejemplo en código Python de este procedimiento en el cuadro 5.4.

Finalización de VLC

Para finalizar la comunicación con el reproductor multimedia VLC, tan solo debemos seguir dos simples pasos:

1. Cerrar la conexión Telnet con VLC mediante la función `close()`.

```
# Creamos una nueva conexión
t.write("new motionjpeg broadcast enabled\r\n")

path = "setup motionjpeg input http://"+parameters["From domain"]+"/video.mjpg\r\n"
#Por ejemplo path = "setup motionjpeg input http://195.243.185.195/mjpg/1/video.mjpg\r\n"

# Cargamos el flujo multimedia
t.write(path)

# Lo reproducimos
t.write("control motionjpeg play\r\n")
```

Cuadro 5.4: Reproducción de vídeo en VLC mediante Telent

2. Cerrar el subproceso VLC mediante la función `terminate()`.

Podemos contemplar un ejemplo en código Python de este procedimiento en el cuadro 5.5.

```
# Cierra la conexión vía Telent
t.close()

# Cierro VLC
vlc.terminate()
```

Cuadro 5.5: Finalización de VLC

5.6. Gestión de usuario y flujos M-JPEG mediante SQLite

En este epígrafe vamos a profundizar en el funcionamiento de la base de datos SQLite utilizada en el sistema receptor y en como interacciona con la aplicación para gestionar las sesiones SIP, establecidas por los usuarios que utilizan el servicio para el envío de flujos M-JPEG.

Las razones de porqué se ha utilizado este elemento, a pesar de que el escenario de pruebas se reduce a un solo terminal, ya fueron detalladas en la sección 2.2.3 del presente documento, por lo que en los siguientes epígrafes nos centraremos en el procedimiento de acceso, consulta y actualización de la base de datos SQLite y su interacción con la aplicación del equipo receptor.

5.6.1. Funcionamiento

En la figura 5.9 podemos examinar un diagrama de flujo simplificado de como gestiona la aplicación las sesiones multimedia con el usuario móvil, almacenando la información de conexión de los flujos M-JPEG emitidos, mediante la base de datos SQLite.

5.6.2. Inicialización de SQLite

La inicialización y puesta en marcha de la base de datos mediante el módulo `sqlite3` es sencilla. Tan solo debemos seguir los siguientes pasos:

1. El objeto `CHservices` que crea la base de datos, debe guardar una referencia al hilo local en su función `_init_()` y almacenar el *db handle*, así como la ruta donde se encuentra la base de datos. Las sentencia necesaria para esto es la siguiente:

   ```
   # Hilo local almacena el "db handle"
   self.local = threading.local()
   ```

2. Almacenar la ruta donde se guardará o donde encuentra la base de datos, por ejemplo, el directorio actual.

   ```
   # Almacenamos la ruta donde se encuentra la base de datos
   self.db_file = "ch_db.dat"
   ```

5.6.3. Establecimiento de conexión con SQLite

Python define varios niveles de seguridad para el acceso a recursos compartidos. En el caso de las bases de datos SQLite, el nivel es 1, lo cual puede comprobarse en el intérprete tecleando: `sqlite.threadsafety`.

Esto implica que aunque los hilos de la aplicación receptora pueden compartir el módulo `sqlite3`, no pueden compartir las conexiones con la base de datos, lo que nos obliga a tener un método `db_open()` para garantizar que abrimos la base de datos desde un hilo cada vez que lo necesitemos.

Para conectar con la base de datos existente, el objeto de `CHservices` de la aplicación receptora utiliza la función `db_open()`, la cual se muestra en el cuadro 5.6.

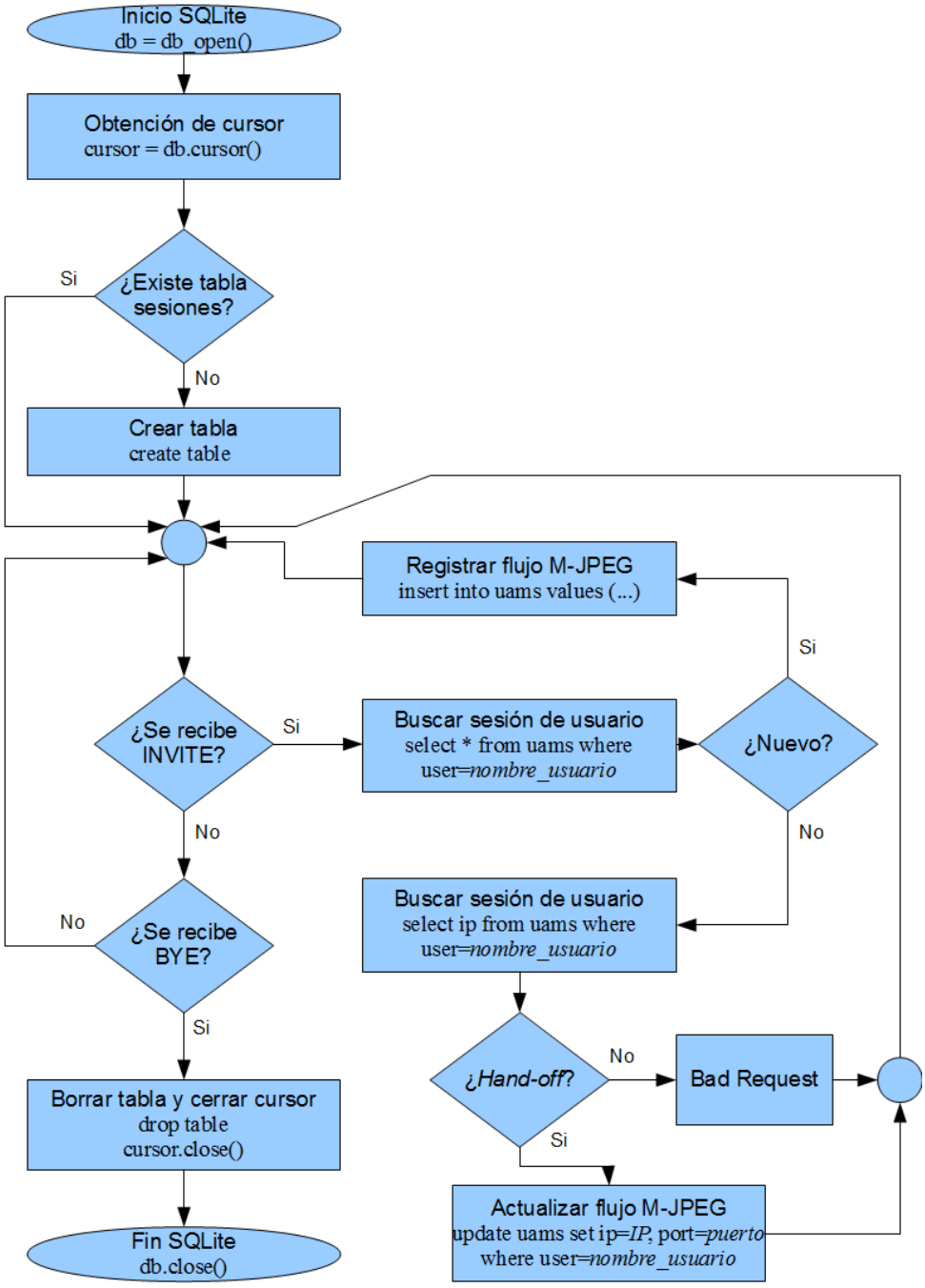

Figura 5.9: Diagrama de flujo de gestión de usuarios en la base de datos del sistema.

```
def db_open(self):
   """Por el nivel de seguridad fijado para los Threads en
   Python, el acceso a las BBDD SQite3 debe tener una conexión
   independiente por cada hilo."""

   if not getattr(self.local, "bbdd", None):
      print "Base de datos SQLite: Conectando..."

      # Una vez que el hilo conecta con la BBDD SQLite3 devuelve
      # la referencia local. Recordemos que cada hilo tiene su
      # conexión con la base de datos por seguridad.
      self.local.db = dbapi.connect(self.db_file)

   return self.local.db
```

Cuadro 5.6: Función db_open()

Como puede observarse, el procedimiento es sencillo. Si el campo `local` contiene una referencia válida, la aplicación conectará con la base de datos cuyo nombre estará contenido en `db_file`.

5.6.4. Acceso y gestión de SQLite

La gestión de la base de datos se basa en el alta, baja o actualización de las sesiones SIP, establecidas por un usuario desde su terminal móvil para el envío de un flujo M-JPEG, y se fundamenta en dos procedimientos esenciales:

1. Creación de un cursor: Cada vez que queramos acceder a la base de datos tendremos que conectar con ella. Esto se hace así por la seguridad establecida en el módulo `threading`. Esto obliga a que las operaciones deban realizarse siempre a través de un cursor.

2. Realización de acciones y consultas mediante el paso de mensajes en formato SQL: La base de datos poseerá diferentes campos donde almacenar la información. En este sentido, el método `execute()` nos permitirá mandar órdenes SQL a la base de datos.

Creación del cursor y la tabla de información de sesiones SIP activas en SQLite

En el caso particular de la aplicación receptora se ha decidido crear una tabla para almacenar los siguientes campos de información de una sesión SIP:

1. Usuario: El campo `user` de la tabla de información contendrá una cadena de texto con el nombre de usuario SIP que ha iniciado una sesión multimedia con el equipo receptor para el envío de un flujo M-JPEG desde su terminal.
2. Dominio: El campo `domain` de la tabla de información contendrá una cadena de texto con el nombre del dominio del usuario SIP que ha iniciado una sesión multimedia con el equipo receptor para el envío de un flujo M-JPEG desde su terminal.
3. Media: El campo `media` de la tabla de información contendrá una cadena de texto con información descriptiva acerca del tipo de flujo enviado por el usuario móvil desde su terminal.
4. IP: El campo `ip` de la tabla de información contendrá una cadena de texto con la dirección IP actual del usuario móvil que ha establecido una sesión multimedia con el equipo receptor para el envío de un flujo M-JPEG desde su terminal.
5. Puerto: El campo `port` de la tabla de información contendrá un número entero con el puerto al que va destinado el flujo M-JPEG enviado desde el terminal móvil al equipo receptor.

En el cuadro 5.7 podemos observar el código fuente necesario para la creación de un cursor que nos permita el acceso a la base de datos, así como la tabla de información referente a las sesión SIP establecida entre el terminal móvil y el equipo receptor.

Insertar o actualizar elementos en SQLite

Si nuestra base de datos soporta transacciones, si estas están activadas, y si la característica de *auto-commit* está desactivada, será necesario llamar al método `commit()` de la conexión para que se lleven a cabo las operaciones definidas en la transacción.

Si en estas circunstancias utilizáramos una herramienta externa para comprobar el contenido de nuestra base de datos sin llamar primero a `commit()`, nos encontraríamos entonces con una base de datos vacía.

```
# Cada vez que queramos acceder a la BBDD tendremos que conectar con ella. Esto se hace por la
# seguridad establecida en el módulo threading.
db = self.db_open()

# Las operaciones se realizan siempre a través de un cursor.
cursor = db.cursor()

# La base da datos tendrá diferentes campos tipo texto donde almacenar la información.
# El método execute() nos permite mandar órdenes SQL a la base de datos.
# Si no existiera, creamos la tabla para gestionar las sesiones SIP
cursor.execute("""create table if not exists uams (user text, domain text, media text, ip text,
 port int)""")
```

Cuadro 5.7: Creación del cursor y la tabla de información de sesión multimedia

Si comprobáramos el contenido de la base de datos desde Python, sin cerrar el cursor ni la conexión, recibiríamos el resultado del contexto de la transacción, por lo que parecería que se han llevado a cabo los cambios, aunque no es así. Por tanto, en este caso, los cambios sólo se aplican al llamar a la función `commit()`.

Para bases de datos que no soporten transacciones, el estándar dicta que debe proporcionarse un método `commit()` con implementación vacía, por lo que no es mala idea llamar siempre a `commit()` aunque no sea necesario, para poder cambiar de sistema de base de datos con solo modificar la línea `import` del principio del fichero Python en cuestión.

Si nuestra base de datos soporta la característica de `rollback` también podemos cancelar la transacción actual con:

```
db.rollback()
```

Tras obtener un cursor, el sistema realiza distintas acciones SQL para insertar o actualizar la información de una sesión SIP establecida con un usuario móvil, así como el flujo M-JPEG recibido:

1. Insertar usuario y flujo M-JPEG nuevo: Al recibirse un mensaje SIP tipo `INVITE`, procedemos a comprobar si la conexión ya estaba establecida, así como imprimir por pantalla el estado actual de la base de datos. Para ello ejecutaremos las sentencias descritas en el cuadro5.8.

2. Actualizar usuario y flujo M-JPEG: Al detectarse un posible *hand-off* o cambio de red por parte de un usuario que ha iniciado un procedimiento SIP del tipo `INVITE-UPDATE`, procedemos a comprobar si la conexión ya estaba establecida para actualizar la información de la sesión. En caso contrario, la

actualización no será aceptada y se le notificará al MS mediante un mensaje SIP de tipo `BAD REQUEST`. Por último imprimimos por pantalla el estado actual de la base de datos. Para ello ejecutaremos las sentencias descritas en el cuadro 5.9.

```
cursor.execute("""insert into uams values (?,?,?,?,?)""", (sip_parameters["From"],
 sip_parameters["From domain"], sdp_parameters["m"], sdp_parameters["o"],
 sdp_parameters["port"]))

db.commit()

# Imprimimos la base de datos para conocer su estado
cursor.execute("""select * from uams""")
print "Base de datos SQLite: Estado actual ->"
for tupla in cursor.fetchall():
print tupla
print " "
```

Cuadro 5.8: Insertar usuario y flujo M-JPEG nuevo

```
cursor.execute("""update uams set ip=?, port=? where user=?""", (sdp_parameters["o"],
 sdp_parameters["port"],sip_parameters["From"],))

db.commit()

# Imprimimos la base de datos para conocer su estado
cursor.execute("""select * from uams""")
print "Base de datos SQLite: Estado actual ->"
for tupla in cursor.fetchall():
print tupla
print " "
```

Cuadro 5.9: Actualizar usuario y flujo M-JPEG

Consultas SQLite

Tras obtener un cursor, el sistema realiza distintas consultas SQL para gestionar y mantener la sesión SIP establecida con un usuario móvil, como por ejemplo:

1. Consulta de identificador de usuario SIP: Al recibirse un mensaje SIP tipo `INVITE`, procedemos a comprobar si la conexión ya estaba establecida, así como si el usuario ha cambiado de red. Posteriormente, se imprime el estado actual de la base de datos. Para ello ejecutaremos las sentencias descritas en el cuadro5.10.

2. Consulta de dirección IP de usuario: Al detectarse un posible *hand-off* o cambio de red por parte de un usuario que ha iniciado un procedimiento SIP del tipo `INVITE-UPDATE`, procedemos a comprobar si la dirección de red del terminal. Para ello ejecutaremos las sentencias descritas en el cuadro 5.11.

```
# Consultamos en la base de datos el identificador SIP de un usuario móvil
cursor.execute("""select * from uams where user=?""", (sip_parameters["From"],))

# Una consulta devuelve una tupla
tuplas=cursor.fetchall()
```

Cuadro 5.10: Consulta SQL de identificador de usuario móvil

```
# Consultamos en la base de datos la dirección IP de un usuario móvil
cursor.execute("""select ip from uams where user=?""", (sip_parameters["From"],))

# Una consulta devuelve una tupla
tuplas=cursor.fetchall()
```

Cuadro 5.11: Consulta SQL de dirección IP de usuario móvil

5.6.5. Finalización y desconexión de SQLite

Si se recibe un mensaje SIP de tipo `BYE`, la sesión establecida entre el terminal móvil y el equipo receptor finaliza. Cuando no existen usuarios conectados al servicio y se desea finalizar la aplicación, se procederá mediante las sentencias descritas en el cuadro 5.12, limpiando la tabla de información de sesiones SIP, cerrando el cursor de referencia, así como la propia base de datos.

```
cursor.execute("""drop table if exists uams""")
cursor.close()
db.close()
print "Base de datos SQLite: Off"
```

Cuadro 5.12: Finalización de SQLite tras recibir un mensaje SIP BYE

5.7. Finalización de sesiones multimedia

En un momento determinado, y en pleno envío de la secuencia de imágenes M-JPEG, el usuario móvil finalizar la sesión multimedia pulsando la opción *Desconectar* del menú principal de la aplicación de su terminal. En ese momento, la función `run()` de `CHservices` notifica a los hilos en ejecución de que el programa debe terminar.

Además, `run()` enviará las respuestas SIP pertinentes tras la recepción de un mensaje `BYE`, con el objetivo de informar al equipo emisor de que la sesión ha finalizado correctamente y la reproducción del flujo M-JPEG ha cesado.

En la figura 4.14 podemos observar el intercambio de mensajes SIP realizado entre el terminal móvil y el equipo receptor para finalizar la sesión multimedia. La secuencia completa de los mensajes intercambiados se detalla en la sección 4.8.

Capítulo 6

Módulos comunes implementados

En este capítulo vamos a describir los diversos módulos implementados que dan soporte a las comunicaciones establecidas entre el terminal móvil y el equipo receptor.

Cada uno de estos módulos está programado con un estilo de programación estructurada, es decir, no están implementados mediante programación orientada a objetos, por lo que sus funciones pueden ser invocadas de forma estática y reutilizadas fácilmente por cualquier otro programa que desarrollemos. En este sentido, tanto el terminal móvil, como el equipo receptor, aprovechan las capacidades proporcionadas por las funciones contenidas en estos módulos.

De forma predeterminada, Python incluye el módulo `socket` para la comunicación entre distintos equipos de red, tanto en la versión para PC como en la versión móvil. Las conexiones establecidas por las aplicaciones de ambos extremos del sistema utilizan este mecanismo para comunicarse. Concretamente, en este proyecto se utilizan *sockets* de tipo datagrama (`socket.SOCK_DGRAM`) que ofrecen un servicio no orientado a conexión basado en el protocolo UDP, pertenecientes a la familia de *sockets* IPv4 (`socket.AF_INET`).

Nuestro diseño hace uso del mecanismo de *sockets* para enviar mensajes SIP y SDP, así como paquetes M-JPEG sobre RTP, entre ambas entidades del sistema. Para ello, se han desarrollado cuatro módulos comunes que permiten generar una serie de mensajes y procesar un grupo concreto de respuestas para estos protocolos.

Para invocar cualquier función de dichos módulos, tan solo es necesario seguir los siguientes pasos:

1. Indicar al principio del fichero la ubicación del módulo en el árbol de direc-

torios del sistema mediante la función `append`. Por ejemplo:

```
import sys
sys.path.append("C:\Symbian\9.2\S60_3rd_FP1\Epoc32\winscw\c\python")
```

2. Importar el módulo deseado al principio de nuestro archivo mediante la sentencia: `import` *nombre del módulo*.

3. Una vez importado, invocar cualquier función mediante la sentencia: `nombre_de_módulo.nombre_de_función()`

6.1. SIPmessages

`SIPmessages.py` es un módulo común del sistema programado en lenguaje Python e implementado en base a la RFC 3261 [24]. Contiene un conjunto de funciones encargadas de generar un grupo de mensajes SIP para ser utilizados en el establecimiento y finalización de sesiones multimedia, así como en otros procedimientos tales como actualización de la información de la conexión, asentimientos, etc.

Este módulo general del sistema ofrece soporte tanto a la aplicación residente en el terminal móvil, como a la del equipo receptor, y su tarea principal consiste en proporcionar las herramientas necesarias para crear e interpretar mensajes SIP.

Las funcionalidades proporcionadas por este módulo se apoyan en el módulo `string` de la librería estándar de Python, el cual proporciona soporte para el tratamiento de cadenas de texto, ya que recordemos que el protocolo SIP utiliza este formato para el envío y recepción de mensajes a través de un *socket* UDP, tanto en el terminal móvil, como en el equipo receptor.

Además de generar las peticiones y respuestas SIP necesarias en el sistema, este módulo también contiene un método, denominado `getParameters()`, que permite procesar un mensaje SIP recibido, devolviendo un diccionario con los parámetros con información significativa encontrados en dicho mensaje.

A diferencia de otras partes del código fuente, este módulo no está orientado a objetos, sino que ha sido desarrollado mediante un estilo de programación estructurada. Está pensado para que pueda ser utilizado como una librería o colección de funciones para futuros desarrollos, y es utilizado tanto por la aplicación instalada en el terminal móvil, como en el equipo receptor, ya que ambos sistemas utilizan el protocolo SIP para establecer y actualizar el estado de sesiones multimedia para la transferencia de información.

Concretamente, el módulo `SIPmessages.py` es capaz de generar una serie de mensajes SIP mediante las siguientes funciones:

- `register`(*fromUser, fromDomain, toUser, toDomain, callId, userAgent*), construye un *string* con un mensaje SIP `REGISTER` para registrarse en el servidor SIP.
- `invite`(*fromUser, fromDomain, toUser, toDomain, callId, userAgent, CSeq, contentLength*), construye un *string* con un mensaje SIP `INVITE` para establecer una sesión multimedia.
- `update`(*fromUser, fromDomain, toUser, toDomain, callId, userAgent*), construye un *string* con un mensaje SIP `UPDATE`.
- `ack`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con un mensaje `ACK` aceptando la conexión SIP.
- `bye`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con un mensaje SIP `BYE` para establecer finalizar una sesión multimedia.
- `cancel`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con un mensaje `CANCEL` para cancelar el último comando SIP.
- `options`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con un mensaje `OPTIONS` con diversa información sobre la sesión multimedia.
- `trying`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con una respuesta 100 en respuesta a un `INVITE` que solicita establecer una sesión multimedia.
- `ringing`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con una respuesta 180 en respuesta a un `INVITE` que solicita establecer una sesión multimedia.
- `sessionProgress`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con una respuesta 183 en respuesta a un `INVITE` que solicita establecer una sesión multimedia.
- `ok`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con una respuesta 200 afirmativa.

- `badRequest`(*fromUser, fromDomain, toUser, toDomain, callId, CSeq, userAgent*), construye un *string* con una respuesta 400 en respuesta a una petición del cliente.

Cada una de las anteriores funciones recibe un conjunto de cadenas de texto por parámetro, que se corresponden respectivamente con los campos SIP: `From`, `From domain`, `To`, `To domain`, `Call-ID`, `CSeq`, `User-Agent` y `Content-Length`.

Por otra parte, el módulo `SIPmessages.py` también proporciona las funcionalidades necesarias para procesar los mensajes SIP recibidos mediante la función:

- `getParameters`(*msg*), se encarga de obtener los parámetros significativos de la sesión de datos establecida por el usuario a partir de las cabeceras contenidas en el mensaje SIP *msg* pasado como parámetro, devolviendo un diccionario `parameters` con los siguientes campos de información:
  - `Request-Line`.
  - `To`.
  - `To domain`.
  - `From`.
  - `From domain`.
  - `Via`.
  - `Call-ID`.
  - `CSeq`.
  - `Contact`.
  - `User-Agent`.
  - `Date`.
  - `Content-Length`.
  - `Content-Type`.
  - `Subject`.

6.2. SDPmessages

`SDPmessages.py` es un módulo común del sistema programado en lenguaje Python e implementado en base a la RFC 4566 [26]. Contiene un conjunto de

funciones encargadas de generar un grupo de mensajes SDP para ser utilizados en el establecimiento y actualización de sesiones multimedia.

Este módulo general del sistema ofrece soporte tanto a la aplicación residente en el terminal móvil, como a la del equipo receptor, y su tarea principal consiste en proporcionar las herramientas necesarias para crear e interpretar mensajes SDP.

Las funcionalidades proporcionadas por este módulo se apoyan en el módulo `string` de la librería estándar de Python, el cual proporciona soporte para el tratamiento de cadenas de texto, ya que recordemos que el protocolo SDP utiliza este formato para el envío y recepción de mensajes a través de un *socket* UDP, en conjunción con los mensajes SIP, tanto en el terminal móvil, como en el equipo receptor.

Además de generar los mensajes con información relevante acerca de la información multimedia transmitida, este módulo también contiene un método, denominado `getParameters()`, que permite procesar un mensaje SDP recibido, devolviendo un diccionario con los parámetros con información significativa encontrados en dicho mensaje.

A diferencia de otras partes del código fuente, este módulo no está orientado a objetos, sino que ha sido desarrollado mediante un estilo de programación estructurada. Está pensado para que pueda ser utilizado como una librería o colección de funciones para futuros desarrollos, y es utilizado tanto por la aplicación instalada en el terminal móvil, como en el equipo receptor, ya que ambos sistemas utilizan el protocolo SDP para fijar los parámetros de conexión y el tipo de contenido multimedia que va a ser transmitido por la red.

Concretamente, el módulo `SDPmessages.py` es capaz de generar una serie de mensajes SDP mediante la función:

- `sdp`(*v, o, s, i, u, e, p, c, b, z, k, a, t, r, m, im, cm, bm, km, am*), construye un *string* con un mensaje SDP, el cual contiene información acerca de la codificación del flujo multimedia que va a ser transmitido, así como diversos parámetros referentes a la conexión.

  - `v`, versión del protocolo.
  - `o`, originador y identificador del protocolo. Aunque se puede utilizar cualquier método generador para identificar la sesión de forma unívoca, se propone el uso del *TimeStamp* de *Network Time Protocol* para asegurar dicha característica a la hora de generar este campo.
  - `s`, nombre de sesión

- `i`, información de sesión. (Opcional).
- `u`, URI de descripción. (Opcional).
- `e`, dirección de correo electrónico. (Opcional).
- `p`, número de teléfono. (Opcional).
- `c`, información de conexión, contiene la IP a la que se enviara el flujo de datos – No requerido si se incluye en todo los medios. (Opcional).
- `b`, cero o más líneas de información de ancho de banda. (Opcional).
- `z`, ajustes de zona horaria. (Opcional).
- `k`, clave de cifrado. (Opcional).
- `a`, cero o más líneas de atributo de sesión. (Opcional).
- `t`, tiempo de sesión activa.
- `r`, cero o más horarios repetidos. (Opcional).
- `m`, nombre multimedia y dirección de transporte.
- `im`, título multimedia. (Opcional).
- `cm`, información de conexión – Opcional si se incluye en el nivel de sesión. (Opcional).
- `bm`, cero o más líneas de información de ancho de banda. (Opcional).
- `km`, clave de cifrado. (Opcional).
- `am`, cero o más líneas de atributo multimedia. (Opcional).

Por otra parte, el módulo `SDPmessages.py` también proporciona las funcionalidades necesarias para procesar los mensajes SIP recibidos mediante la función:

- `getParameters`(*msg*), se encarga de obtener los parámetros significativos, referentes al flujo multimedia transmitido, del mensaje SDP recibido, extrayendo la información a partir de las cabeceras contenidas en dicho mensaje *msg* pasado como parámetro, devolviendo un diccionario `parameters` con los siguientes campos de información:
  - `v`, versión de protocolo.
  - `o`, originador del flujo multimedia.
  - `s`, nombre de sesión.
  - `c`, información de conexión.
  - `t`, tiempo de sesión activa.
  - `m`, nombre multimedia y dirección de transporte.
  - `port`, puerto de la conexión donde se encuentra disponible el flujo M-JPEG enviado por el dispositivo móvil.

6.3. RTPpackets

`RTPpackets.py` es un módulo común del sistema programado en lenguaje Python e implementado en base a la RFC 3551 [27]. Contiene un conjunto de funciones encargadas de generar las cabeceras de los paquetes RTP para ser utilizados como contenedores de fragmentos de una secuencia o flujo de imágenes en formato M-JPEG.

Este módulo general del sistema ofrece soporte tanto a la aplicación residente en el terminal móvil, como a la del equipo receptor, y su tarea principal consiste en proporcionar las herramientas necesarias para el envío fragmentado de imágenes, así como su reensamblado en destino mediante el protocolo RTP.

Las funcionalidades proporcionadas por este módulo se apoyan en el módulo `struct` de la librería estándar de Python, el cual proporciona soporte para el tratamiento de octetos y su correcta composición para el envío por la red, ya que recordemos que el protocolo RTP utiliza este formato en la transmisión y recepción de datagramas a través de un *socket* UDP, tanto en el terminal móvil, como en el equipo receptor.

El módulo estándar `struct` proporciona dos funciones básicas para el empaquetado y desempaquetado de octetos denominadas: `struct.pack(...)` y `struct.unpack(...)`. Además, permite la correcta ordenación de los bytes para su envío por la red mediante el modificador exclamación (`!`). Las funciones desarrolladas en el módulo `RTPpackets` hacen uso de estas herramientas, proporcionadas por el módulo estándar `struct`, para generar correctamente los octetos de los paquetes RTP.

Por otro lado, este módulo también cuenta con un método, denominado `getParameters(...)`, que permite procesar un datagrama RTP recibido, devolviendo un tupla con los parámetros significativos encontrados en dicho paquete.

A diferencia de otras partes del código fuente, este módulo no está orientado a objetos, sino que ha sido desarrollado mediante un estilo de programación estructurada. Está pensado para que pueda ser utilizado como una librería o colección de funciones para futuros desarrollos, y es utilizado tanto por la aplicación instalada en el terminal móvil, como en el equipo receptor, ya que ambos sistemas utilizan el protocolo RTP para enviar fragmentos de información de la secuencia de imágenes M-JPEG.

Concretamente, el módulo `RTPpackets.py` es capaz de generar los paquetes RTP mediante la función:

- `getRTPheader`(*Ve, P, X, CC, M, payloadType, sequenceNumber, timestamp,*

SSRC), genera la cabecera de un paquete RTP a partir de los parámetros recibidos.

- `Ve` (2 bits), versión utilizada del protocolo RTP (por defecto v. 2).
- `P` (1 bit), es un bit que indica si el paquete lleva relleno. Si `P=1`, el último byte del relleno indica cuantos octetos hay que ignorar (por defecto vale cero).
- `X` (1 bit). Si `X=1`, hay una cabecera de extensión al final de la normal. No se usa.
- `CC` (4 bits), el número de mezcladores indican de cuantas fuentes se están abasteciendo el sistema (por defecto vale cero).
- `M` (1 bits). El bit marcador tiene distinta funcionalidad dependiendo del tipo de datos transportados. En este sistema indica el fin de una imagen.
- `payloadType` (7 bits), indica el tipo de datos, el contenido (audio, vídeo) y su codificación (*codec* empleado, cifrado, etc.). En este sistema tendrá el valor `payloadType=26` (M-JPEG).
- `sequenceNumber` (16 bits), el número de secuencia identifica cada fuente con un número seleccionado aleatoriamente. Se incrementa en uno con cada paquete enviado.
- `tiemstamp` (32 bits), la marca de tiempo.
- `SSRC` (32 bits), generado aleatoriamente por la fuente, es un identificador único de una fuente para cada sesión.

Por otra parte, el módulo `RTPpackets.py` también proporciona las funcionalidades necesarias para procesar los paquetes RTP recibidos mediante la función:

- `getParameters`(*msg, packet_size*), se encarga de obtener los parámetros significativos del paquete RTP a partir de las cabeceras contenidas en el datagrama *msg* pasado como parámetro, devolviendo una tupla con los siguientes elementos:
  - `parameters`, contiene una tupla con los siguientes elementos de información:
    - `Ve - P - X - CC` (2, 1, 1 y 4 bits respectivamente), octeto que contiene varios parámetros RTP: versión (`Ve`) utilizada en el protocolo RTP, bit de *padding* (`P`) que indica si el paquete lleva octetos de relleno, bit *eXtension* que indica si existen cabeceras de extensión al final del datagrama RTP y campo *Countributing Count*

(`CC`) para indicar el número de mezcladores o fuentes que abastecen el sistema.

- `M - payloadType` (1 y 7 bits respectivamente), octeto que contiene varios parámetros RTP: bit *Mark* o marcador para el indicar el final de un *frame* o imagen, y campo *Payload Type*, que indica el tipo de datos contenido en la sección de datos, ya sea audio o vídeo, y su tipo de codificación (en nuestro caso M-JPEG tipo 26)
- `sequenceNumber` (16 bits), contiene el número de secuencia y debe incrementarse en una unidad con cada paquete enviado
- `timestamp` (32 bits), contiene el *Tiemstamp* y debe comenzar en cero.
- `SSRC` (32 bits), *Synchronization Source Identifier* contiene un número generado aleatoriamente por la fuente y representa un identificador único de una fuente para cada sesión.

6.4. MJPEGpackets

`MJPEGpackets.py` es un módulo común del sistema programado en lenguaje Python e implementado en base a la RFC 2435 [28]. Contiene un conjunto de funciones encargadas de generar las cabeceras de los paquetes M-JPEG que precederán a los fragmentos de una secuencia o flujo de imágenes en formato M-JPEG.

Las funcionalidades proporcionadas por este módulo se apoyan en el módulo `struct` de la librería estándar de Python, el cual proporciona soporte para el tratamiento de octetos y su correcta composición para el envío por la red, ya que recordemos que M-JPEG utiliza este formato en la transmisión y recepción de datagramas a través de RTP, tanto en el terminal móvil, como en el equipo receptor.

El módulo estándar `struct` proporciona dos funciones básicas para el empaquetado y desempaquetado de octetos denominadas: `struct.pack(...)` y `struct.unpack(...)`. Además, permite la correcta ordenación de los bytes para su envío por la red mediante el modificador exclamación (`!`). Concretamente, las función desarrollada en el módulo `MJPEGpackets` hace uso de la función `struct.pack(...)` para generar correctamente los octetos de las cabeceras de los paquetes M-JPEG.

Por otro lado, este módulo también cuenta con un método, denominado `getParameters(...)`, que permite procesar un datagrama M-JPEG recibido, devolviendo un tupla con los parámetros significativos encontrados en dicho paquete

y los datos transportados.

A diferencia de otras partes del código fuente, este módulo no está orientado a objetos, si no que ha sido desarrollado mediante un estilo de programación estructurada. Está pensado para que pueda ser utilizado como una librería o colección de funciones para futuros desarrollos, y es utilizado tanto por la aplicación instalada en el terminal móvil, como en el equipo receptor, ya que ambos sistemas utilizan el formato M-JPEG para enviar fragmentos de información de la secuencia de imágenes.

Concretamente, el módulo `MJPEGpackets.py` es capaz de generar los paquetes M-JPEG mediante la función:

- `getJPEGheader`(*typeSpecific, fragmentOffset, typeJPEG, q, width, height*), genera la cabecera de un paquete M-JPEG a partir de los parámetros recibidos.
  - `typeSpecific` (8 bits), su interpretación depende del valor del tipo de campo. Si está a cero se ignora en recepción.
  - `fragmentOffset` (24 bits), representa el *offset* en bytes del paquete actual dentro de un *frame* de datos JPEG. Este valor se codifica siguiendo la ordenación en bytes de la red, es decir, el byte más significativo primero. El valor de este campo nunca debe exceder los 2^{24} bytes. El primer paquete tendrá *offset* 0.
  - `typeJPEG` (8 bits), transporta información que de otra forma sería representada en la tabla de especificaciones JPEG como parámetros de estilo adicionales JFIF.
  - `q` (8 bits), este campo define la cuantización de las tablas para el *frame* actual. 0-127 indica que la cuantización se realiza utilizando el algoritmo especificado en `Type`. Valores entre 128-255 indican que la cuantización aparece especificada en una cabecera que aparece justo después de la cabecera principal JPEG.
  - `width` (8 bits), codifica la anchura de imagen en múltiplos de 8 pixels. Por ejemplo, `width=40` equivale a una imagen de anchura 320 *pixels*. El valor máximo son 2040 *pixels*.
  - `height` (8 bits), codifica la altura de imagen en múltiplos de 8 pixels. Por ejemplo, `height=30` equivale a una imagen de anchura 240 *pixels*. El valor máximo son 2040 *pixels*.

Por otra parte, el módulo `MJPEGpackets.py` también proporciona las funcionalidades necesarias para procesar los paquetes M-JPEG recibidos mediante la función:

- `getParameters`(*msg, packet_size*), se encarga de obtener los parámetros significativos del paquete M-JPEG a partir de las cabeceras contenidas en el datagrama *msg* pasado como parámetro, devolviendo una tupla con los siguientes elementos:

  1. Parámetros M-JPEG:

     - `typeSpecific - fragmentOffset` (8 y 24 bits respectivamente). *Type Specific* es un campo cuya interpretación depende del valor del campo tipo. Si está a cero se ignora en recepción. *Fragment Offset* representa el *offeset* en octetos del paquete actual dentro de un *frame* de datos JPEG. Este valor se codifica siguiendo la ordenación de la red, es decir, el byte más significativo primero. El valor de este campo nunca debe exceder los 2^{24} bytes. El primer paquete tendrá *offset* 0.
     - `typeJPEG` (8 bits), transporta información que de otra forma sería representada en la tabla de especificaciones JPEG como parámetros de estilo adicionales JFIF.
     - `q` (8 bits). Este campo define la cuantización de las tablas para el *frame* actual.
     - `width` (8 bits), codifica la anchura de imagen en múltiplos de 8 *pixels*.
     - `height` (8 bits), codifica la altura de imagen en múltiplos de 8 *pixels*.

  2. `data`, contiene los datos del paquete.

6.5. Otros módulos y utilidades

Debido a que la plataforma PyS60 posee ciertas particularidades con respecto a la versión de Python para computadora, ha sido necesario desarrollar un módulo adicional denominado `I2B`, a modo de utilidad adicional para su uso en la aplicación móvil.

Mientras Python para PC soporta de forma predeterminada funciones como `bin(...)`, incluida por defecto en el módulo `__builtin__`, que permiten el paso de números enteros(*integer* o *long*) a cadenas binarias de texto, en PyS60 esto no es posible, por lo que ha sido necesario desarrollar un módulo adicional que proporcione dichas utilidades.

6.5.1. I2B

`I2B` proporciona dos funciones Python para el paso de números a formato binario y viceversa:

- `bin2int`(b), recibe un número binario codificado en forma de cadena de caracteres y devuelve un número entero.
- `int2bin`(i), recibe un número entero codificado en formato *integer* y devuelve un su equivalente binario en forma de cadena de texto.

Capítulo 7

Pruebas

En este capítulo se presenta la batería de pruebas realiza al sistema.

En primer lugar, se ha comprobado el correcto funcionamiento de los módulos comunes implementados, ya que estos elementos son utilizados tanto por la aplicación móvil, como por la aplicación receptora.

Posteriormente se ha comprobado que, tanto el software desarrollado para el terminal móvil, como el software desarrollado para el equipo receptor, cumplen los requisitos de diseño, permitiendo la gestión de sesiones SIP para la movilidad en Internet, así como el envío y reproducción de flujos M-JPEG.

En la mayoría de las pruebas realizadas se ha utilizado una herramienta software de auditoría de red denominada Wireshark, la cual ha permitido el análisis minucioso de los protocolos implementados, el contenido de los paquetes RTP enviados, así como los mensajes SIP intercambiados por el sistema.

7.1. Pruebas de los módulos comunes del sistema

7.1.1. Pruebas *SIPmessages*

Las pruebas realizadas para este módulo, así como los resultados obtenidos, se resumen en la tabla 7.1.

Prueba realizada	Resultado	Observaciones

Mensaje `REGISTER`.	OK	Aunque este mensaje al final no es utilizado por el sistema, ya que el presente trabajo se ciñe a el establecimiento de una sesión multimedia entre dos extremos, se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Mensaje `INVITE`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado. Tras finalizar este documento se detectó un fallo en la generación del campo `CSeq` (que contiene la petición del mensaje de solicitud y el número de secuencia) en los procedimientos de *hand-off* o cambio de red. Este problema fue correctamente subsanado mediante la inclusión de una variable contador en la aplicación móvil `MS`.
Mensaje `UPDATE`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `ACK`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Mensaje `BYE`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Mensaje `CANCEL`.	OK	Aunque esta respuesta al final no es utilizada por el sistema, se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `100 Trying`.	OK	Aunque esta respuesta al final no es utilizada por el sistema, se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `180 Ringing`.	OK	Aunque esta respuesta al final no es utilizada por el sistema, se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `183 Session progress`.	OK	Aunque esta respuesta al final no es utilizada por el sistema, se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `200 Ok`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.
Respuesta `400 Bad request`.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado.

Cuadro 7.1: Pruebas módulo *SIPmessages*

Por otro lado, se ha comprado que la función `getParameters`(*msg*) del módulo

SIPmessages devuelve, a partir de un mensaje SIP pasado por parámetro, un diccionario formado por los siguientes elementos:

- `Request-Line`.
- `To`.
- `To domain`.
- `From`.
- `From domain`.
- `Via`.
- `Call-ID`.
- `CSeq`.
- `Contact`.
- `User-Agent`.
- `Date`.
- `Content-Length`.
- `Content-Type`.
- `Subject`.

7.1.2. Pruebas *SDPmessages*

Las pruebas realizadas para este módulo, así como los resultados obtenidos, se resumen en la tabla 7.2.

Prueba realizada	Resultado	Observaciones

Campos de descripción de sesión.	OK	Se ha comprobado que los diversos campos implementados posean una estructura adecuada dentro del mensaje enviado. Entre los parámetros de descripción disponibles, tan solo se han probado los campos obligatorios utilizados por el sistema: ▪ `v` ▪ `o` ▪ `s` ▪ `c`
Descripción horaria.	OK	Entre los parámetros de descripción horaria disponible, tan solo se han probado los campos obligatorios utilizados por el sistema: ▪ `t`
Descripción multimedia.	OK	Entre los parámetros de descripción multimedia disponible, tan solo se han probado los campos obligatorios utilizados por el sistema: ▪ `m`

Cuadro 7.2: Pruebas módulo *SDPmessages*

Por otro lado, se ha comprado que la función `getParameters`(*msg*) del módulo SDPmessages devuelve, a partir de un mensaje pasado por parámetro, un diccionario formado por los siguientes elementos:

- `v`, versión del protocolo.
- `o`, originador y identificador del protocolo. Aunque se propone el uso del *TimeStamp* de *Network Time Protocol* para generar el identificador de dicho campo, realmente puede utilizarse cualquier otro método, por lo que en nuestro caso particular hemos utilizado el reloj del sistema.
- `s`, nombre de sesión.
- `c`, información de conexión, contiene la IP a la que se enviará el flujo de datos.
- `t`, tiempo de sesión activa.
- `m`, nombre multimedia y dirección de transporte.

- `port`, puerto donde se encuentra disponible el flujo enviado por el dispositivo móvil.

7.1.3. Pruebas *RTPpackets*

Las pruebas realizadas para este módulo, así como los resultados obtenidos, se resumen en la tabla 7.3.

Prueba realizada	Resultado	Observaciones
Versión de protocolo.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen 2 bits con la versión (2) del protocolo.
Bit `P`.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen 1 bit de relleno con valor `P=0` por defecto.
Bit `X`.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen 1 bit de extensión con valor `X=0` por defecto.
Cuenta de contribuyentes.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen un campo `Contributing Count` de 4 bits con valor `CC=0` por defecto.
Bit `M`.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen 1 bit `M` con valor `M=0` por defecto.
Tipo de carga.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen un campo `Payload Type` de 7 bits con valor `Payload Type=26` por defecto, el cual indica que los datos transportados se corresponden con un flujo en formato M-JPEG.
Número de secuencia.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen un campo `Sequence Number` de 16 bits con valor `Sequence Number=0` por defecto.
Marca de tiempo.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen un campo `Timestamp` de 32 bits con valor `Timestamp=0` por defecto.
Identificador de fuente de sincronización.	OK	Se ha comprobado que los diversos paquetes RTP generados mediante la función `getRTPheader()` contienen un campo `Synchronization Source Identifier` de 32 bits con valor `SSRC=0` por defecto.

Identificador de fuente de contribución.	No OK	Debido a que el sistema utiliza una única fuente al mismo tiempo para generar el flujo M-JPEG, este campo no se ha implementado.

Cuadro 7.3: Pruebas módulo *RTPpackets*

Por otro lado, se ha comprado que la función `getParameters`(*msg, packet_size*) del módulo `RTPpackets` devuelve, a partir de un paquete recibido pasado por parámetro, una tupla formada por los siguientes elementos:

- `Ve - P - X - CC` (2, 1, 1 y 4 bits respectivamente), octeto que contiene varios parámetros RTP. Versión (`Ve`) utilizada en el protocolo RTP, bit de *padding* (`P`),bit `X` que indica si existen cabeceras de extensión y el campo *Countributing Count* (`CC`).
- `M - payloadType` (1 y 7 bits respectivamente), octeto que contiene varios parámetros RTP. bit `M` o marcador de fin de imagen, y campo *Payload Type* (en nuestro caso M-JPEG tipo 26).
- `sequenceNumber` (16 bits), contiene el número de secuencia.
- `timestamp` (32 bits), contiene el *Tiemstamp*.
- `SSRC` (32 bits), *Synchronization Source Identifier* contiene un identificador generado aleatoriamente por la fuente.

7.1.4. Pruebas *MJPEGpackets*

Las pruebas realizadas para este módulo, así como los resultados obtenidos, se resumen en la tabla 7.4.

Prueba realizada	Resultado	Observaciones
Tipo específico.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Type-specific` de 8 bits con valor `Type-specific=0` por defecto. Su interpretación depende del valor del tipo de campo. Si está a cero se ignora en recepción.
Desplazamiento de fragmento.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Fragment Offset` de 24 bits. Dicho campo representa el desplazamiento bytes del fragmento actual dentro de una imagen JPEG.

Tipo.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Type` de 8 bits.
Cuantización.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Quantization` de 8 bits con valor `Q=0` por defecto.
Anchura de las imágenes.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Width` de 8 bits con valor `Width=30` por defecto, equivalente a 240 *pixels*.
Altura de las imágenes.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados mediante la función `getJPEGheader()` contienen un campo `Height` de 8 bits con valor `Height=40` por defecto, equivalente a 320 *pixels*.

Cuadro 7.4: Pruebas módulo *MJPEGpackets*

Por otro lado, se ha comprado que la función `getParameters`(*msg, packet_size*) del módulo `MJPEGpackets` devuelve, a partir de un paquete recibido pasado por parámetro, una tupla formada por los siguientes elementos:

- `typeSpecific - fragmentOffset` (8 y 24 bits respectivamente). *Type Specific* es un campo cuya interpretación depende del valor del campo tipo. Si está a cero se ignora en recepción. *Fragment Offset* representa el *offeset* en octetos del paquete actual dentro de un *frame* de datos JPEG.
- `typeJPEG` (8 bits).
- `q` (8 bits).
- `width` (8 bits), codifica la anchura de imagen en múltiplos de 8 *pixels*.
- `height` (8 bits), codifica la altura de imagen en múltiplos de 8 *pixels*.
- `data`, contiene los datos del paquete.

7.2. Pruebas en el emulador software del terminal móvil

Las pruebas realizadas a la aplicación móvil, así como los resultados obtenidos, se resumen en la tabla 7.5.

Prueba realizada	Resultado	Observaciones
Establecimiento de sesiones multimedia.	OK	Se ha comprobado que la aplicación móvil es capaz de establecer sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Actualización de sesiones multimedia y simulación de *hand-off* o cambio de red.	OK	Se ha comprobado que la aplicación móvil es capaz de actualizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Finalización de sesiones multimedia.	OK	Se ha comprobado que la aplicación móvil es capaz de finalizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Bit `P` de relleno RTP.	OK	Se ha comprobado que los paquetes RTP generados por la aplicación móvil contienen un campo `P=1` cuando los datos que transportan contiene *padding* o relleno al final de los mismos. En estos casos, el último bytes de datos indica cuantos octetos tienen que ser ignorados.
Bit `M` marcador RTP.	OK	Se ha comprobado que los paquetes RTP generados por la aplicación móvil contienen un campo `M=1` cuando los datos que transportan contiene el fragmento final de una imagen JPEG completa, perteneciente a una secuencia M-JPEG.
Número de secuencia RTP.	OK	Se ha comprobado que los paquetes RTP generados por la aplicación móvil contienen un campo `Sequence Number` que se incrementa en una unidad con cada fragmento enviado, y se reinicia cuando alcanza el valor `Squence Number=65535`.
Desplazamiento de fragmento JPEG.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados por la aplicación móvil contienen un campo `Fragment Offset` que se incrementa acorde con el número de octetos enviados, y se reinicia cuando alcanza el valor `Fragment Offset=`2^{24}.
Finalización de hilos de la aplicación.	OK	Se ha comprobado que los diversos hilos de ejecución lanzados por la aplicación para el envío de mensajes SIP, así como el envío de flujos M-JPEG finalizan correctamente mediante condición de parada o invocación a `thread.exit_thread()`.

Cuadro 7.5: Pruebas aplicación móvil en el emulador software

7.3. Pruebas en el equipo receptor

Las pruebas realizadas a la aplicación receptora, así como los resultados obtenidos, se resumen en la tabla 7.6.

Prueba realizada	Resultado	Observaciones
Establecimiento de sesiones multimedia.	OK	Se ha comprobado que la aplicación receptora es capaz de establecer sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Actualización de sesiones multimedia y simulación de *hand-off* o cambio de red.	OK	Se ha comprobado que la aplicación receptora es capaz de actualizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Finalización de sesiones multimedia.	OK	Se ha comprobado que la aplicación receptora es capaz de finalizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Bit `P` de relleno RTP.	OK	Se ha comprobado que los paquetes RTP con campo `P=1` son tratados adecuadamente, descartando el número de octetos de relleno indicados en el último bytes de datos.
Bit `M` marcador RTP.	OK	Se ha comprobado que los paquetes RTP con campo `M=1` transportan el fragmento final de una imagen JPEG completa, por lo que deben ser reproducidos.
Número de secuencia RTP.	OK	Se ha comprobado que los paquetes RTP generados por la aplicación móvil contienen un campo `Sequence Number` que se incrementa en una unidad con cada fragmento enviado, y se reinicia cuando alcanza el valor `Squence Number=65535`.
Desplazamiento de fragmento JPEG.	OK	Se ha comprobado que los diversos paquetes M-JPEG generados por la aplicación móvil contienen un campo `Fragment Offset` que se incrementa acorde con el número de octetos enviados, y se reinicia cuando alcanza el valor `Fragment Offset=`2^{24}.
Comportamiento ante pérdida de paquetes.	OK	Se ha comprobado que ante la pérdida de fragmentos pertenecientes a una imagen JPEG completa, el equipo receptor descarta dicha imagen para reproducir la imagen de la secuencia inmediatamente posterior.
Imagen corrupta.	OK	Se ha comprobado que si la imagen recibida no contiene un formato adecuado para su reproducción, la aplicación receptora captura la excepción generada al intentar visualizar dicha imagen, la descarta, y procede a reproducir la imagen de la secuencia inmediatamente posterior.
Finalización de hilos de la aplicación.	OK	Se ha comprobado que los diversos hilos de ejecución lanzados por la aplicación para el intercambio de mensajes SIP, así como la reproducción de flujos M-JPEG finalizan correctamente mediante condición de parada o invocación a `thread.exit_thread()`.

Cuadro 7.6: Pruebas de la aplicación del equipo receptor.

7.4. Pruebas de integración e interoperabilidad

En la tabla 7.7 se resume la batería de pruebas realizada al sistema completo para verificar la integración de cada uno de los subsistemas así como su interoperabilidad.

Prueba realizada	Resultado	Observaciones
Establecimiento de sesiones multimedia.	OK	Se ha comprobado que el sistema es capaz de establecer sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Actualización de sesiones multimedia y simulación de *hand-off* o cambio de red.	OK	Se ha comprobado que el sistema es capaz de actualizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Finalización de sesiones multimedia.	OK	Se ha comprobado que el sistema es capaz de finalizar sesiones multimedia mediante un correcto intercambio de mensajes SIP.
Secuencia de comando SIP.	OK	Se ha comprobado que los mensajes SIP generados por la aplicación móvil y la aplicación receptora contienen la petición del mensaje de solicitud y el número de secuencia adecuado según el contexto del intercambio.
Tamaño de paquete RTP 16 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 16 bytes.
Tamaño de paquete RTP 20 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 20 bytes.
Tamaño de paquete RTP 24 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 24 bytes.
Tamaño de paquete RTP 32 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 32 bytes.
Tamaño de paquete RTP 64 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 64 bytes.
Tamaño de paquete RTP 128 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 128 bytes.
Tamaño de paquete RTP 184 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 184 bytes.
Tamaño de paquete RTP 256 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 256 bytes.
Tamaño de paquete RTP 384 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 384 bytes.

Tamaño de paquete RTP 448 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 448 bytes.
Tamaño de paquete RTP 1024 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 1024 bytes.
Tamaño de paquete RTP 1460 octetos.	OK	Se ha comprobado que el sistema es capaz enviar y reproducir flujos con tamaño de paquete RTP de 1460 bytes, caso límite de carga para una red Ethernet. En este caso, y aunque la ejecución es correcta, se detecta una leve saturación en la parte receptora a la hora de reproducir por pantalla la secuencia de imágenes recibida.
Cierre abrupto de la aplicación móvil.	OK	Si en mitad del envío y reproducción de un flujo M-JPEG, la aplicación móvil se cierra abruptamente mediante la tecla `Exit` del terminal, tanto la aplicación móvil, como la aplicación receptora, finalizan correctamente. Todos los hilos del sistema finalizan correctamente su ejecución. Cierre ordenado de la conexión (*sockets*) y finalización de la sesión SIP establecida.
Cierre abrupto de la aplicación receptora.	No OK	Si en mitad del envío y reproducción de un flujo M-JPEG, la aplicación receptora se cierra abruptamente, el sistema no finaliza correctamente y la aplicación móvil mostrará un mensaje de error (*System Error (-50)*) en el intérprete de Python para S60. Esto se debe a que este caso no se ha contemplado en este diseño al no representar uno de los objetivos principales del presente trabajo. Este problema podría solucionarse asignando la lógica necesaria al botón *Salir* de la interfaz gráfica de usuario de la aplicación receptora.

Cuadro 7.7: Pruebas del sistema completo

Capítulo 8

Historia del proyecto

Este proyecto surge a partir de un trabajo de investigación realizado en GAST (Grupo de Aplicaciones y Servicios Telemáticos), en el Departamento de Ingeniería Telemática de la Universidad Carlos III de Madrid, durante los meses de Noviembre de 2009 a Octubre de 2010. Pero su verdadero inicio se remonta a la realización de un estudio tecnológico previo, durante los meses de Febrero a Septiembre de 2009, donde se revisó el estado actual de la plataforma PyS60.

Durante los meses de Noviembre de 2009 a Octubre de 2010 ha habido distintos períodos de desarrollo, cuya explicación es objeto de este apartado. En este sentido, durante este capítulo trataremos de explicar las diferentes etapas del proyecto, comentando los objetivos y el trabajo llevado a cabo en cada una de ellas, así como los distintos problemas que han ido surgiendo a lo largo del camino y los nuevos enfoques que tuvieron que ser formulados para solventarlos.

En la figura 8.1 puede observarse un diagrama de Gantt simplificado, con la duración de cada una de las fases desarrolladas durante este trabajo. Dichas fases se detallen en los epígrafes posteriores del presente apartado.

8.1. FASE I: Planteamiento inicial y definición de requisitos

8.1.1. Descripción de las tareas realizadas

Una de las primeras tareas del presente proyecto, consistía en comprobar la viabilidad de utilizar el lenguaje de programación Python para crear aplicaciones para terminales S60, para poder abordar posteriormente la implementación

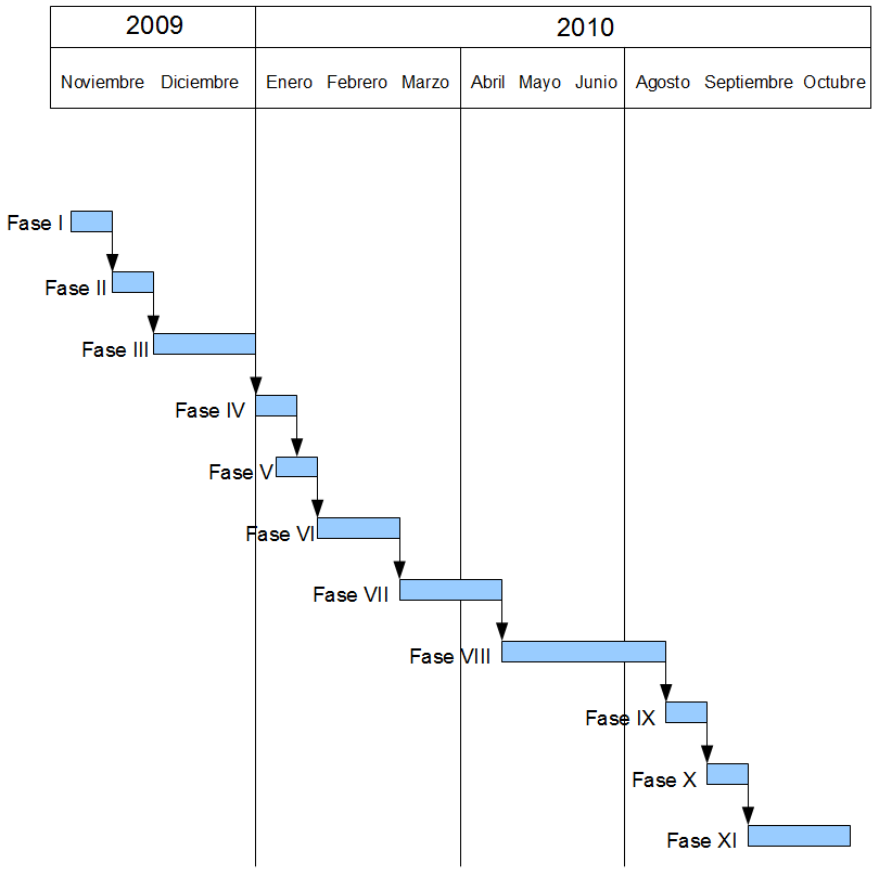

Figura 8.1: Diagrama de Gantt simplificado de las fases de desarrollo del proyecto

de una aplicación multimedia que ofreciera movilidad en Internet mediante el protocolo SIP.

En esta línea de trabajo, se procede a elegir las herramientas software que se utilizarán para implementar dicho sistema, así como a desarrollar una documentación inicial para su correcta instalación. En este sentido se procedió a poner en marcha la plataforma PyS60 sobre el emulador, proporcionado por Nokia mediante el paquete software SDK *S60 Developers Tools* para terminales de 3ª Edición FP1, así como fijar los requisitos iniciales del diseño.

La duración de esta tarea fue de dos semana aproximadamente.

8.1.2. Resultados

Como resultado de esta primera fase de estudio, se obtuvieron las siguientes conclusiones:

1. Se desarrollará de una aplicación multimedia para terminales Symbian S60, que permita la movilidad en Internet mediante el protocolo SIP. Para ello se implementará un sistema cuyo funcionamiento esté basado en el borrador del IETF "Soporte para la movilidad multimedia con SIP" [23].

2. La aplicación práctica deberá desarrollarse exclusivamente mediante el lenguaje de programación Python. El software implementado para el dispositivo móvil hará uso de los módulos PyS60, específicos para la serie 60 de terminales celulares.

3. En el sistema participarán dos tipos de entidades o usuarios. Los encargados de reproducción y visualización de los datos multimedia, y los usuarios móviles, responsables del envío de los mismos. Los datos enviados serán vídeo u otro tipo de contenido multimedia.

4. La visualización de la información se llevará acabo mediante algún tipo de reproductor multimedia.

8.2. FASE II: Desarrollo del módulo SIP

8.2.1. Descripción de las tareas realizadas

La primera tarea del proyecto tenía como objetivo desarrollar un módulo común del sistema que fuera capaz de crear e interpretar una colección de men-

sajes SIP. Dicha colección debería permitir a dos entidades establecer, actualizar y finalizar sesiones multimedia.

Durante esta etapa se repasaron los conceptos principales de la señalización SIP mediante la RFC 3261, a la vez que se profundizó en el estudio del borrador del ITEF "Soporte para la movilidad multimedia con SIP".

La duración de esta tarea fue de aproximadamente dos semanas.

8.2.2. Resultados

Como resultado de esta fase se implementó el módulo Python `SIPmessages.py`, cuyas funcionalidades se han descrito en el capítulo 6 del presente documento.

8.3. FASE III: Desarrollo de un prototipo para el envío y reproducción de imágenes JPEG sobre HTTP/TCP

8.3.1. Descripción de las tareas realizadas

Se decide realizar el envío de una secuencia de imágenes JPEG mediante TCP/HTTP para su posterior reproducción en VLC, con el objetivo de comprobar el grado de compatibilidad del software implementado con un reproductor comercial. Para ello, se decide desarrollar el código fuente necesario para generar mensajes `multipart/x-mixed-replace` sobre HTTP.

La duración de esta tarea fue de un mes aproximadamente.

8.3.2. Resultados

Tras la prueba de este sistema, se decide migrar hacia una solución más elaborada, basada en los protocolos RTP y UDP. Esta decisión de diseño se basa en la idea de que TCP y HTTP no resultan una buena combinación en un servicio para la reproducción de imágenes en directo, una situación que se agrava por el hecho de enmarcarse en un contexto de redes móviles, donde las pérdidas o las variaciones en el retardo pueden resultar significativamente mayores que un ámbito de red fija.

Cuando TCP y HTTP sufren un error de transmisión, siguen intentando transmitir los paquetes de datos perdidos, hasta conseguir una confirmación de que la información llegó en su totalidad. Sin embargo, un servicio no orientado a conexión como UDP, continúa mandando datos de forma ininterrumpida. De este modo, el sistema implementado mediante este esquema ofrece una respuesta más tolerante frente a situaciones de pérdidas, descartándose los paquetes necesarios para continuar con la reproducción.

Se procede a documentar el trabajo realizado, acción materializada en las secciones 4.6.2 y 5.5.2 del presente proyecto.

8.4. FASE IV: Actualización de los requisitos del diseño

8.4.1. Descripción de las tareas realizadas

A partir de este momento, se procedió a realizar un análisis más detallado en donde tener en cuenta diferentes aspectos como:

- Complejidad del código que se iba a desarrollar.
- Funcionalidad e interfaces a implementar.
- Versiones de los protocolos soportados por el sistema.
- Integración de los subsistemas subyacentes.
- Codificación M-JPEG.
- Gestión de sesiones activas.

La duración de esta tarea fue de dos semanas aproximadamente.

8.4.2. Resultados

Se toma la decisión de implementar tres nuevos módulos para la creación e interpretación de información SDP, así como el envío de datos M-JPEG sobre RTP.

8.5. FASE V: Desarrollo del módulo SDP

8.5.1. Descripción de las tareas realizadas

La implementación del sistema basado en el protocolo RTP necesita de la creación de un nuevo módulo, que sea capaz de generar e interpretar la información de codificación de los datos multimedia intercambiados entre el terminal móvil y el equipo receptor.

En este sentido, se desarrolla un módulo común del sistema capaz de interpretar una colección de mensajes SDP basados en la RFC 4566.

La duración de esta tarea fue de dos semanas aproximadamente.

8.5.2. Resultados

Como resultado de esta fase, se implementó el módulo Python `SDPmessages.py`, cuyas funcionalidades se han descrito en el capítulo 6 del presente documento.

8.6. FASE VI: Desarrollo de los módulos RTP y M-JPEG

8.6.1. Descripción de las tareas realizadas

Con el objetivo de transportar la información enviada por el terminal móvil hasta el equipo receptor, fue necesario dos módulos comunes del sistema para la creación e interpretación de las cabeceras RTP y M-JPEG.

En este sentido, se decide diseñar dos módulos capaces de crear e interpretar datagramas RTP y M-JPEG, basados en la RFC 3551 y RFC 2435 respectivamente.

La duración de esta tarea fue de aproximadamente un mes.

8.6.2. Resultados

Como resultado de esta fase se implementaron los módulos Python `RTPpackets.py` y `MJPEGpackets`, cuyas funcionalidades se han descrito en el

capítulo 6 del presente documento.

8.7. FASE VII: Desarrollo de una aplicación móvil con soporte para la movilidad SIP para el envío de flujos M-JPEG sobre RTP

8.7.1. Descripción de las tareas realizadas

Se diseña un primer prototipo de la aplicación para el terminal móvil del sistema, utilizándose los módulos comunes del sistema, así como las librerías proporcionadas por PyS60.

Se realizan las primeras pruebas de funcionamiento para el envío de imágenes mediante el formato M-JPEG, procediéndose a la corrección de errores.

La duración de esta tarea fue dos mes aproximadamente.

8.7.2. Resultados

1. Ciertas incompatibilidades entre el núcleo de Python para PC y el núcleo de Python de PyS60 obligan a la creación de un módulo denominado `I2B`.

2. Ciertas incompatibilidades entre el núcleo de Python para PC y el núcleo de Python de PyS60 obligan a la utilización del módulo estándar `thread` en lugar de `threading`, lo que implica algunos cambios en el diseño de la aplicación móvil original.

3. Se crea la aplicación `MS.py`, cuya funcionalidad se describe en el capítulo 4 del presente documento.

8.8. FASE VIII: Desarrollo de una aplicación receptora con soporte para la movilidad SIP para la reproducción de flujos M-JPEG sobre RTP

8.8.1. Descripción de las tareas realizadas

Se diseña un primer prototipo de la aplicación para el equipo receptor, utilizándose los módulos comunes del sistema, así como las librerías estándar de Python.

Se realizan las primeras pruebas de funcionamiento para la reproducción de flujos M-JPEG, procediéndose a la corrección de errores.

La duración de esta tarea fue de dos meses y medio aproximadamente.

8.8.2. Resultados

1. Se decide diseñar un reproductor propietario mediante la librería estándar `Tkinter` y la librería Python externa `PIL`.

2. Se crea la aplicación `CH.py` y `CHservices.py`, cuya funcionalidad se describe en el capítulo 5 del presente documento.

8.9. FASE IX: Estudio de extensiones Symbian C++ para PyS60

8.9.1. Descripción de las tareas realizadas

Como tarea paralela, también se estudió durante el proyecto la posibilidad crear una extensión Symbian C++ para un terminal S60. Este estudio se prolongó durante dos semanas, al final de las cuales se documentó un prototipo, a modo de ejemplo, capaz de modificar y mostrar un texto por pantalla sobre una de las teclas de navegación del terminal.

Aunque esta tarea no era uno de los objetivos del proyecto, pretende dejar la puerta abierta a futuros desarrollos que posibiliten el tratamiento directo de vídeo mediante un módulo Python basado en una extensión C++ a tal efecto.

8.9.2. Resultados

Realización de una extensión denominada `uikludges.py` que sirva de ejemplo básico para mostrar como crear, compilar y construir una extensión. Dicho módulo ofrece la posibilidad de poder renombrar la etiqueta `Exit-key` en PyS60.

En la sección D se documenta el procedimiento para crear, compilar y firmar dicha extensión, así como los enlaces pertinentes para descargar el código fuente original del ejemplo.

8.10. FASE X: Pruebas

8.10.1. Descripción de las tareas realizadas

Se aplicó una batería de pruebas para comprobar el funcionamiento de la aplicación desarrollada así como los módulos comunes implementados.

La duración de esta tarea fue de dos semanas aproximadamente.

8.10.2. Resultados

1. Pruebas de los módulos comunes `SIPmessages`, `SDPmessages`, `RTPpackets` y `MJPEGpackets` para verificar su correcto funcionamiento, así como la generación e interpretación de paquetes.

2. Pruebas de integración de los subsistemas adyacentes de la aplicación `MS`.

3. Pruebas de integración de los subsistemas adyacentes de la aplicación `CH` y `CHservices`.

4. Pruebas generales de funcionamiento del sistema completo.

5. Corrección de errores.

6. Documentación del código fuente.

Para mayor detalle de las pruebas realizadas, se recomienda consultar el capítulo 7 del presente documento.

8.11. FASE XI: Documentación

8.11.1. Descripción de las tareas realizadas

En esta última fase se realizaron varias tareas:

1. Documentar el sistema implementado, así como su funcionamiento y puesta en marcha.
2. Generación de una serie de manuales para instalar correctamente las herramientas de trabajo en un equipo y en un terminal móvil.
3. Documentación de las pruebas realizadas.

La duración de esta tarea fue un mes y medio.

8.11.2. Resultados

Esta fase consiste en la documentación de todo el trabajo realizado, acción materializada en la presente memoria.

Capítulo 9

Conclusiones y trabajos futuros

9.1. Conclusiones

En los últimos años, la denominada "revolución móvil" ha propiciado un despegue sin precedentes en el ámbito de las aplicaciones para terminales celulares.

Estamos asistiendo a una etapa de popularización de la conectividad a redes inalámbricas. Los dispositivos móviles nos permiten comunicarnos casi desde cualquier lugar, y el rápido desarrollo de estas tecnologías ha desembocado en la incorporación de nuevas funcionalidades en los terminales: GPS, cámara fotográfica, acelerómetro, etc.

El objetivo principal de este proyecto, consistía en crear una aplicación multimedia para terminales Symbian de la serie 60, que permitiera la movilidad en Internet mediante el protocolo SIP. Dentro de este contexto, uno de los requisitos de diseño necesarios para crear dicho sistema, era la utilización del lenguaje de programación Python y el conjunto de librerías PyS60, ya que que se deseaba realizar un estudio paralelo de las capacidades ofrecidas por esta tecnología para la creación y puesta en marcha de aplicaciones en el entorno móvil, y más concretamente, en la amplia gama de terminales S60.

La gran cantidad de módulos Python y PyS60 disponibles, transfieren a este lenguaje un gran potencial para el desarrollo de aplicaciones móviles capaces de aprovechar las nuevas características ofrecidas por los terminales actuales.

En este sentido, este proyecto ha planteado una posible implementación de un sistema para el envío de flujos M-JPEG sobre RTP, desde un terminal móvil S60 a un equipo fijo, una arquitectura que pretende explotar las nuevas capacidades y potencia de cómputo de esta generación de dispositivos móviles, así como soportar

las necesidades ubicuas actuales mediante los protocolos SIP y SDP.

En lo que respecta a la aplicación móvil desarrollada, y como hemos podido comprobar a lo largo del presente estudio, trabajar con PyS60 ofrece tres ventajas fundamentales a la hora de implementar nuestras aplicaciones móviles:

1. Los programas son compactos y permite acceder a características multimedia del teléfono (acceso a la cámara fotográfica, *Bluetooth*, conexiones GPRS, información de localización, etc.).

2. Facilita la creación de *widgets* nativos, por lo que resulta extremadamente sencillo crear interfaces gráficas de usuario.

3. Python y PyS60 incorporan una gran cantidad de módulos con funciones ya implementadas, lo que facilita en gran medida el desarrollo de nuevas aplicaciones.

La aplicación móvil aprovecha parte de las funcionalidades ofrecidas por PyS60 para implementar un sistema capaz de enviar un flujo de imágenes, capturadas por la cámara integrada del dispositivo, o previamente almacenadas en memoria, en el caso de un emulador software.

Aunque en un principio se pensó capturar y enviar un vídeo "en vivo" mediante la cámara integrada del terminal móvil, durante el estudio exhaustivo de las características actuales de PyS60 se decidió desestimar esta idea, ya que, si bien existe un módulo capaz de ofrecer las características necesarias para la captura y almacenamiento de vídeo, este conjunto de librerías todavía no cuenta con las funcionalidades suficientes para su manipulación directa durante su adquisición.

Este hecho, motivó una revisión de los requisitos iniciales del sistema, decidiéndose enviar en su lugar un flujo M-JPEG. Aun así, se ha pensado en la posibilidad de desarrollar, como futuro trabajo, una extensión Symbian C++ para PyS60 que permita la manipulación y envío de vídeo en directo. En este sentido, se propone el apéndice introductorio D del presente documento.

Respecto a la aplicación implementada en el equipo receptor, se ha creado un sistema reproductor capaz de visualizar por pantalla los flujos M-JPEG enviados desde un terminal móvil, permitiendo la reproducción continua de los mismos aunque el terminal realice un *hand-off*, o cambio de red, en un momento determinado.

Aunque en un principio se pensó utilizar un reproductor multimedia como VLC, para visualizar el flujo M-JPEG recibido en el equipo receptor, finalmente se optó por desarrollar un reproductor propio basado en tecnología Python,

las librerías estándar TkInter y la librería Python externa PIL. Esta decisión se tomó en base a que, según la documentación oficial del equipo VideoLAN, VLC todavía no implementa completamente la RFC 2453 [28], lo que dificulta la realización de pruebas en modo servidor con el reproductor, para examinar el formato de los datagramas esperados por la aplicación. Por esta razón, y a pesar de que las pruebas realizadas mediante el analizador de protocolos Wireshark arrojaban resultados correctos, en lo que a formación de paquetes se refiere, no fue posible la reproducción del flujo enviado por el terminal móvil mediante VLC.

Por otra parte, ha sido preciso implementar toda la lógica adicional para gestionar las sesiones de envío de datos multimedia, para lo que fue necesario crear dos agentes de usuario SIP, capaces de actualizar, según el contexto, la información de una sesión en curso entre el terminal móvil y el equipo receptor.

En este sentido, durante este trabajo se han desarrollado una serie de módulos comunes, a modo de utilidades generales, para la creación e interpretación de mensajes SIP y SDP, así como las cabeceras de los paquetes RTP y M-JPEG. Dichos módulos son partes reaprovechables del código fuente, siendo utilizadas tanto por la aplicación implementada para el terminal móvil, como la aplicación desarrollada para el equipo receptor.

Tras este estudio se concluye que, Python y PyS60 resultan una opción a tener en cuenta a la hora de desarrollar aplicaciones móviles, ya que nos han permitido implementar un sistema multimedia capaz de ofrecer soporte para la movilidad en Internet mediante el protocolo SIP.

9.2. Líneas futuras

A continuación se proponen una serie de mejoras, así como trabajos futuros, que podrían considerarse a la hora de desarrollar nuevas aplicaciones multimedia para la plataforma S60, mediante la utilización del lenguaje de programación Python y el conjunto de módulos PyS60 para terminales móviles de la serie 60.

9.2.1. Envío de vídeo desde el terminal móvil

Uno de los inconvenientes de utilizar un lenguaje de programación "no nativo" para el desarrollo de la aplicación móvil de este sistema, en lugar del lenguaje Symbian C++, consiste en que PyS60 todavía no está dotado de las capacidades necesarias para acceder a algunas funcionalidades de bajo nivel del terminal.

Este es el caso de las capacidades vídeo, donde PyS60 proporciona dos

funciones a través del módulo `camera`, `start_record`(*filename, callable*) y `stop_record()`, que si bien permiten la captura y almacenamiento de vídeos en la memoria del terminal, para su posterior envío o reproducción, no ofrecen la posibilidad de acceder a dicho flujo de vídeo en tiempo real para su tratamiento.

En Symbian C++ se pueden usar funciones de `CMdaAudioPlayerUtility` (como `OpenFileL()` o `OpenDesL()`) para la reproducción de contenidos multimedia, pero se necesita un mecanismo independiente para la recuperación de los archivos en el dispositivo.

Un posible trabajo futuro consistiría en crear una extensión Python en C++, que pueda ser invocada y manejada mediante las funciones de un módulo PyS60, y que permita acceder de forma nativa a las capacidades vídeo del terminal.

Este desarrollo permitiría plantear en un futuro un sistema más complejo, basado en un esquema de videoconferencia bidireccional entre dispositivos.

Para más detalles de como crear, compilar y firmar una extensión C++ para PyS60, se recomienda consultar el apéndice D del presente documento.

9.2.2. Pruebas en un entorno real

El presente proyecto se centra en los extremos de la conexión, en este caso un terminal móvil enviando información, y un equipo receptor capaz de reproducir y visualizar los datos multimedia recibidos. Pero en un caso real, existe una tercera entidad, denominada *proxy SIP*, que se encarga de registrar a los usuarios, así como de determinar su ubicación y su disponibilidad *online*.

El módulo `SIPmessages`, desarrollado durante este trabajo, ofrece la posibilidad de crear e interpretar mensajes `REGISTER`, por lo que, implementando la lógica necesaria, podría plantearse un sistema más complejo que incluyera una tercera entidad para registrar a los usuarios en el servicio. En este sentido, podría resultar interesante realizar pruebas con algún sistema real como OpenSER.

Otro aspecto importante a tratar es la seguridad, ya que el sistema implementado tan solo se basa en la identificación SIP del usuario, así como la dirección y puerto de conexión. Esta información, enviada en claro por la red, resulta vulnerable a un posible atacante que desee manipular la reproducción del flujo M-JPEG enviado. Por este motivo, sería necesario dotar al sistema de extensiones de seguridad SIP apropiadas.

Por último, cabe destacar que el sistema implementado durante este trabajo, ha sido probado en un entorno local, mediante la utilización de un emulador software S60. En este sentido, sería deseable ajustar y probar el sistema en un

terminal móvil real Symbian S60.

9.2.3. Crear una aplicación Python ejecutable

Otra posible mejora consistiría en agrupar los ficheros de la aplicación móvil en un programa Python.

Aunque durante este documento se ha revisado como ejecutar *scripts* Python en nuestro terminal móvil, mediante la instalación y posterior utilización del intérprete Python S60, lo más habitual, suele ser que el programa que diseñemos se lance directamente al pulsar sobre su icono en nuestro menú de aplicaciones.

Si deseamos conseguir este tipo de comportamiento en nuestras aplicaciones, necesitaremos utilizar una herramienta denominada `py2sis`, la cual genera archivos `*.sis`, los cuales son ficheros instalables en un sistema Symbian.

Este tipo de aplicaciones, las cuales pueden ser ejecutadas de forma autónoma, solo funcionarán si Python para S60 se encuentra instalado en nuestro dispositivo, aunque existe la posibilidad de incluir el intérprete de Python para S60 en nuestro paquete `*.sis`.

De forma genérica los pasos para crear nuestra propia aplicación serían los siguientes:

1. Descargar e instalar S60 SDK para Symbian OS en nuestra computadora (este procedimiento se explica en el apéndice B del presente documento, para una instalación en un entorno Microsoft Windows). Asegurarse de que las utilidades `makesis` y `uidcrc` estén correctamente configuradas en la variable `PATH` del sistema.

2. Instalar el *plug-in* de Python, el cual viene incluido en el paquete SDK de Python para S60 (este procedimiento se explica en el apéndice B del presente documento).

3. Abrir una ventana de sistema.

4. Utilizar el comando:

   ```
   py2sis <src> [sisfile] [--uid=0x1234567]
   ```

   - `<src>` es la ruta donde se encuentra el *script* Python que deseamos empaquetar.
   - `[sisfile]` es el nombre del nuevo fichero `*.sis` que será generado.

- `[--uid=0x1234567]`: es un UID que puede elegirse en un rango entre 0x01000000 y 0x0fffffff.

Por ejemplo, podemos ejecutar:

```
py2sis mi\_script\_python.py mi\_primer\_programa.sis --uid=0x0FFFFFF
```

Por último tenemos que destacar algunos aspectos relacionados con la seguridad de la plataforma Symbian. Dependiendo del modelo de terminal Symbian que tengamos, podremos instalar directamente o no una aplicación generada de este modo. Por ejemplo, si nuestro terminal, para el cual estamos desarrollando la aplicación, pertenece a la 2ª Edición o anterior, el fichero `*.sis` que generemos con la herramienta `py2sis` podrá ser instalado directamente.

Por el contrario, si el terminal pertenece a la 3ª Edición, como es el caso que estamos analizando en el presente documento, será necesario realizar una posterior certificación de la aplicación generada. Este hecho se debe a que la plataforma de seguridad Symbian gestiona el acceso a determinadas APIs mediante un modelo de capacidades.

Apéndice A

Presupuesto

En este capítulo presentamos el presupuesto asociado al desarrollo del presente proyecto, cuya duración abarca el periodo comprendido entre noviembre del año 2009 a Octubre de 2010.

En primer lugar, se detallan los costes de personal en que se ha incurrido y, a continuación, se muestran los costes derivados del material empleado. La adición de cada uno de estos costes constituirá el presupuesto final.

Los costes totales incluyen los honorarios del Ingeniero de Telecomunicación encargado del desarrollo del proyecto, así como los materiales empleados durante la realización del mismo, entre los que se encuentran:

- Un ordenador portátil Samsung modelo R530-JS03ES, con sistema operativo Microsoft Windows, valorado en 430,83 euros (sin IVA).
- Salvo el sistema operativo utilizado para desarrollar este trabajo, el software utilizado en su totalidad es de libre distribución.
- Conexión a Internet durante la realización del proyecto. Ha sido necesaria para conseguir documentación y descargar los paquetes software necesarios para la realización del sistema, sus pruebas posteriores, así como el texto del presente documento. Está valorada aproximadamente en 39 euros al mes, que multiplicado por los once meses de trabajo, supone un coste de 429 euros.

Cabe destacar que, el presente presupuesto no incluye los gastos de compra de un terminal móvil S60, debido a que el desarrollo, puesta en marcha, y prueba del sistema, se ha realizado con ayuda de un emulador software.

En la siguiente página se incluye el presupuesto total desglosado en una plantilla proporcionada por la Escuela Politécnica Superior de la Universidad Carlos III de Madrid. En ella, se detalla que el presupuesto final para la realización de este proyecto asciende a **36.176 euros**.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:

Álvarez Vallejo, Víctor

2.- Departamento:

Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo: **Aplicaciones PyS60 y movilidad SIP en terminales S60**
- Duración (meses): **11**

Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

40.000,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación [a] (hombres mes)	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
		Ingeniero Senior		4.289,54	0,00	
Álvarez Vallejo, Víctor		Ingeniero	11	2.694,39	29.638,29	
					0,00	
					0,00	
		Hombres mes 11		**Total**	29.638,29	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Portátil Samsung R530-JS03ES	430,83	100	11	60	78,99
		100		60	0,00
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	78,99

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Conexión a Internet	Movistar	429,00
	Total	429,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	29.638
Amortización	79
Subcontratación de tareas	0
Costes de funcionamiento	429
Costes Indirectos	6.029
Total	36.176

Apéndice B

Manual de instalación

B.1. Instalación en PC

B.1.1. Python para PC

A continuación, vamos a describir brevemente el proceso de instalación de Python para sistemas Microsoft Windows, por ser este, el sistema operativo de mayor implantación. La instalación en otras plataformas como GNU/Linux, es todavía más sencilla, puesto que, al ser un programa de código abierto, suele encontrarse en la mayoría de repositorios de las distribuciones más populares. Lo primero que debemos hacer para instalar Python es descargar la versión adecuada de su página web oficial (ver Cuadro B.1).

Accedemos a la web oficial de Python: `http://www.python.org` A continuación, entraremos en la sección de descargas (*DOWNLOAD*) y bajaremos el instalador de IDLE de versión adecuada para nuestro desarrollo, así como el tipo de sistema operativo que tengamos instalado en nuestra computadora.

Cuadro B.1: Sitio web oficial Python

El paquete auto-instalable descargado contiene dos elementos fundamentales: El intérprete de Python para Windows (que nos permitirá ejecutar los programas que implementemos en dicho lenguaje de programación) y la herramienta de desarrollo IDLE, que nos permitirá disponer de un entorno sencillo para implementar programas Python con facilidad (ver Figura B.1).

Puesto que el desarrollo posterior para la serie S60 se basará en la utilización

de un núcleo Python 2.x, en el presente documento hemos optado por descargar la versión 2.6, la cual ofrece compatibilidad con toda la rama 2.x. Hemos tomado esta decisión porque las versiones 3.x implementan cambios más profundos respecto las versiones 2.x, lo que hace existan ciertas incompatibilidades entre dichas ramas.

Una vez instalado IDLE, podremos lanzar el intérprete de Python, así como el editor de programas desde nuestro menú: Inicio ¿Todos los programas ¿Python 2.6 ¿IDLE (Python GUI).

Entorno de desarrollo IDLE

Aunque los programas Python podrían desarrollarse en una simple aplicación tipo *bloc* de notas, por comodidad, se recomienda al lector instalar un entorno de desarrollo integrado adecuado.

Un entorno de desarrollo integrado o IDE (en inglés, *Integrated Development Environment*), es un programa compuesto por un conjunto de herramientas útiles para el desarrollador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

En general, un IDE es un entorno de programación que ha sido empaquetado como una aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como Python, C, C++, C#, Java, Delphi, Visual Basic, etc.

Como ya hemos dicho, es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante *plug-ins* se le puede añadir soporte para lenguajes adicionales.

Durante este trabajo se ha decidió utilizar el entorno de desarrollo integral oficial de Python, denominado IDLE [18] (ver figura B.1), el cual está escrito completamente en este lenguaje, constando de un intérprete en modo consola (para ejecutar nuestros *scripts* Python) y un editor de programas.

IDLE proporciona varias características interesantes como:

- IDLE es un entorno muy sencillo y poco pesado.
- IDLE está disponible para multitud de plataformas (Windows, Mac, Linux).
- IDLE es un editor multi-ventana, con funciones de auto-sangrado del código y auto-completado del mismo.

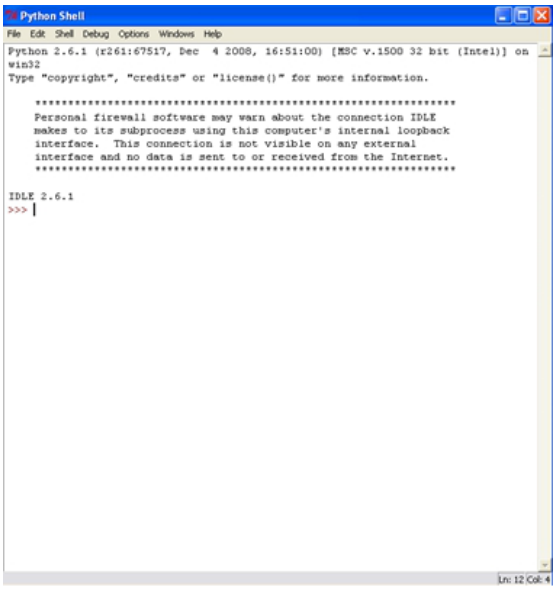

Figura B.1: IDLE Python *Shell* 2.6.1.

- Resulta un entorno de desarrollo suficiente para implementar aplicaciones de relativa complejidad.

El desarrollador es libre de elegir cualquier otro tipo de entorno de desarrollo más específico como: Eclipse, NetBeans, Eric, etc., con los *plug-ins* Python adecuados.

En cualquier caso, para desarrollar y probar nuestros programas Python se necesitan tres elementos diferenciados: Un entorno de desarrollo integrado (que nos sirva para implementar nuestros programas en Python), el intérprete de Python (para ejecutar los scripts realizados) y un teléfono móvil S60 o en su defecto un emulador software (véase Figura B.2).

B.1.2. *Python Imaging Library* (PIL)

PIL es una librería que proporciona las funciones necesarias para permitir la manipulación de imágenes en formato JPEG desde una interfaz TkInter. El módulo PIL está disponible para sistemas Windows, Mac y Linux, y proporciona una librería externa para el lenguaje de programación Python que añade soporte para la apertura y guardado de imágenes en diferentes formatos, soportando versiones Python desde la 2.2 a la 2.6.

La instalación de PIL en Microsoft Windows se puede llevar a cabo mediante

Figura B.2: Software involucrado en la creación de aplicaciones PyS60.

Python Imaging Library (PIL) :::

Commercial Support
Old Releases
Software License
Frequently Asked Questions

Python Imaging Library (PIL)

The **Python Imaging Library (PIL)** adds image processing capabilities to your Python interpreter. This library supports many file formats, and provides powerful image processing and graphics capabilities.

Status

The current free version is PIL 1.1.7. This release supports Python 1.5.2 and newer, including 2.5 and 2.6. A version of 1.1.7 for 3.X will be released later.

Support

Free Support: If you don't have a support contract, please send your question to the Python Image SIG mailing list. The same applies for bug reports and patches.

You can join the Image SIG via python.org's subscription page, or by sending a mail to image-sig-request@python.org. Put *subscribe* in the message body to automatically subscribe to the list, or *help* to get additional information.

You can also ask on the Python mailing list, *python-list@python.org*, or the newsgroup *comp.lang.python*. **Do not send support questions to PythonWare addresses**.

Commercial Support: If you're using PIL in commercial applications, or mission-critical internal applications, you may purchase a PythonWare PIL support package. Contact info@pythonware.com for details.

Downloads

The following downloads are currently available:

PIL 1.1.7

- **Python Imaging Library 1.1.7 Source Kit** (all platforms) (November 15, 2009)
- **Python Imaging Library 1.1.7 for Python 2.4** (Windows only)
- **Python Imaging Library 1.1.7 for Python 2.5** (Windows only)
- **Python Imaging Library 1.1.7 for Python 2.6** (Windows only)

Figura B.3: PIL 1.1.7 para sistemas Microsoft Windows.

unos sencillos pasos:

1. Descarga la aplicación de instalación para Windows desde el sitio web de PIL `http://www.pythonware.com/products/pil/`.

2. Doble *click* en el archivo descargado para iniciar el proceso de instalación.

3. Eligir el directorio adecuado para la instalación de PIL.

4. Completar la instalación.

Concretamente, en este proyecto se ha utilizado la versión de PIL 1.1.7 para entornos Microsoft Windows.

B.1.3. Instalación de la aplicación en el equipo receptor

El usuario deberá almacenar los archivos pertenecientes a la aplicación receptora en una misma carpeta de su sistema. Dichos archivos son:

- `CH.py`
- `CHservices.py`
- `SIPmessages.py`
- `SDPmessages.py`
- `RTPpackets.py`
- `MJPEGpackets.py`
- `ch_logo.gif`

Si desea distribuir los archivos en diferentes carpetas deberá indicarlo al comienzo de los ficheros `CH.py` y `CHservices.py` mediante las instrucción:

```
import sys

# Ruta de ejemplo:
sys.path.append("C:\Symbian\9.2\S60_3rd_FP1\Epoc32\winscw\c\python")
```

B.2. Instalación en terminal móvil

B.2.1. Python para S60

Para probar nuestras aplicaciones Python para S60 tenemos dos opciones. Podemos utilizar directamente un terminal móvil de la serie 60 al que le hayamos instalado un intérprete Python para S60. Por otro lado, también podemos optar por probar nuestros programas en nuestro PC mediante un emulador software de la serie 60.

En los siguientes epígrafes vamos a detallar como descargar e instalar ambas alternativas, siguiendo para ello la documentación oficial [11].

Instalación del software en un teléfono móvil

Para empezar, en este epígrafe, instalaremos el intérprete de software de Python para S60 en nuestro teléfono. Mediante esta instalación dispondremos de un entorno adecuado para ejecutar aplicaciones Python en nuestro móvil, así como gran cantidad de módulos para el desarrollo de nuevas aplicaciones, alojados en librerías estándar.

Para instalar el intérprete Python en nuestro teléfono seguiremos los siguientes pasos:

1. Descargar el intérprete software Python para S60 en nuestro ordenador desde el proyecto *SourceForge's* PyS60. Para ello accedemos a la web `http://sourceforge.net/projects/pys60`. El instalador que necesitamos está disponible como un fichero `.SIS` (archivo de instalación Symbian). Es importante tener en cuenta que será preciso descargar el archivo correcto basándonos en el modelo de nuestro teléfono (por ejemplo, conocer si nuestro modelo se basa en la 2ª o 3ª edición de la plataforma de desarrollo S60).

2. A continuación instalaremos el intérprete de Python para S60 en nuestro teléfono móvil mediante *Bluetooth* o utilizando un cable USB. Para esta tarea podemos utilizar el software denominado *PC Suite*, proporcionado por Nokia.

La figura B.4 muestra una captura del software Nokia PC Suite, así como sus diversas opciones.

Figura B.4: Nokia PC Suite 7.1 para Windows XP/Vista/7.

Después de dicho proceso, nuestro teléfono estará listo para ejecutar los *scripts* que desarrollemos en Python. Para ello, debería aparecer en el menú de nuestro móvil el icono de Python, mediante el cual acceder al intérprete y a la posibilidad de ejecutar programas escritos en este lenguaje. Para más información se recomienda consultar B.2.

Puesto que el procedimiento anteriormente descrito se actualiza con frecuencia, puede ser recomendable visitar la documentación detallada sobre PyS60:
`http://wiki.pensource.nokia.com/projects/Python_for_S60`
`http://www.mobilenin.com/pys60/menu.htm`
`http://mobildevices.kom.aau.dk/index.php?id=909`

Cuadro B.2: Tutoriales en línea sobre PyS60

Instalación del software en un emulador

En este apartado vamos a describir como instalar la plataforma de emulación de dispositivos de la serie S60 en nuestro ordenador. Para ello, vamos a intentar describir todos los elementos que el lector necesita conocer para realizar una instalación correcta de S60 SDK para Symbian OS (conocido de forma abreviada como "S60 SDK").

Mediante esta plataforma dispondremos de un entorno de emulación software donde poder probar nuestros programas Python, antes de cargarlos en el terminal móvil.

Algunos requisitos hardware y software que debemos tener en cuenta antes de proceder a realizar los pasos que se describen en los párrafos siguientes son:

1. Se recomienda instalar el presente software en un equipo con un procesador de 1 GHz, 512 MB de RAM, 1 GB de espacio libre en disco, resolución 1024x768, color 16-bit, tarjeta de sonido compatible con el soporte de audio Microsoft Windows, teclado y ratón. Otro tipo de periféricos como tarjetas de red, dispositivos infrarrojos o *Bluetooth* están soportados. En caso de duda acerca de los modelos concretos soportados se recomienda consultar la documentación adjunta que se proporciona en el paquete de software descargado.

2. Este procedimiento es válido para los sistemas operativos Microsoft Windows XP SP2 (o superior) y Microsoft Windows 2000 SP4.

3. La instalación debe ser realizada utilizando una cuenta de administrador. En caso contrario algunas variables del entorno podrían no ser configuradas correctamente.

4. El SDK debe ser instalado en una ruta de nuestro disco duro que no contenga caracteres de espacio en blanco.

Este apartado pretende describir la instalación de dos componentes necesarios e imprescindibles para probar y ejecutar nuestros *scripts* Python para terminales móviles S60 en nuestro ordenador:

1. Descarga e instalación del emulador software para S60: La primera tarea que vamos a realizar consistirá en la descarga e instalación de un emulador S60, el cual nos permitirá simular el comportamiento de dichos terminales móviles en nuestro ordenador.

2. Descarga e instalación del intérprete Python para S60: El segundo paso consistirá en la descarga e instalación del intérprete Python, el cual, como ya comentamos anteriormente, no suele venir instalado por defecto en el terminal (en este caso un emulador software instalado en nuestro PC). Después de dicha instalación, nuestro emulador S60 dispondrá de un icono de acceso al intérprete Python en su menú de aplicaciones instaladas, con lo que ya podremos empezar a probar nuestros *scripts* Python para S60.

Para empezar vamos a descargar el *kit* de desarrollo S60 SDK. En nuestro caso particular vamos a proceder a descargar e instalar una versión concreta de este *kit*: S60 3ª Edición *Feature Pack* 1 SDK para Symbian OS C++.

S60 SDK para Symbian OS permite desarrollar aplicaciones para dispositivos basados en la plataforma S60 usando C++. Este *kit* de desarrollo se basa en el sistema operativo Symbian, y más concretamente, en esta tercera edición, en la versión 9.2 de este sistema.

El SDK incluye todas las funcionalidades necesarias para el desarrollo de aplicaciones: documentación, API, herramientas *add-on*, compilador, y lo más importante de todo, un emulador para S60.

El único elemento que se echa de menos en este *kit* es un IDE para el desarrollo de nuestros *scripts*, pero como ya se comentó en epígrafe B.1.1, existen multitud de entornos de descarga gratuita a disposición del desarrollador. En concreto, en

el presente documento hemos comentado alguno específico y sencillo de utilizar como IDLE para el desarrollo de *scripts* en lenguaje Python, pero se puede utilizar otros de uso más general como Eclipse (con *plug-ins* adecuados como `Pydev`).

Para instalar S60 SDK y el intérprete Python para S60 en nuestro PC deberemos seguir los siguientes pasos:

1. Descargar (de manera gratuita) e instalar en nuestro PC S60 *Platform SDK for Symbian OS*, para C++, la cual incluye el dispositivo emulador, que nos permitirá probar y ejecutar las aplicaciones que implementemos para dicha serie de terminales. Para ello debemos dirigirnos a la página web de Forum Nokia `http://forum.nokia.com` y registrarnos, con el objetivo de tener pleno acceso a las aplicaciones de desarrollo disponibles.

2. Una vez hecho esto, procederemos a la descarga del paquete auto-instalable. Si aun no estamos registrados, o si deseamos consultar otra documentación complementaria deberemos visitar `http://forum.nokia.com`. Por otra parte, si ya estamos registrados, podemos descargar directamente el instalador en el siguiente enlace: `http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html`

   En nuestro caso, como ya comentamos anteriormente, hemos optado por descargar la versión S60-SDK 3ª Edición FP1 (*Feature Pack* 1). Es importante recordar, que si deseamos instalar la 3ª Edición, o alguna anterior, es necesario tener instalado previamente en nuestro ordenador las últimas versiones de JRE (*Java Runtime Enviroment*) así como Active Perl para el correcto funcionamiento.

3. Descargar e instalar Java *Runtime* (versión `1.4.2_02` o superior) en la web: `http://www.java.com/en/download/manual.jsp`.

4. Descargar e instalar Active Perl (versión 5.6.1 o superior) en la web: `http://www.activestate.com/activeperl/`

   Tanto Active Perl como JRE son necesarios para poder utilizar Sisar, AIF *Builder*, CS *Help Compiler*, así como poder acceder a las preferencias del emulador.

5. Una vez instalados ambos paquetes de software gratuitos, procederemos a instalar nuestra plataforma S60 SDK para Symbian OS anteriormente descargada.

6. En el último paso de la instalación de S60 SDK para Symbian OS, se nos preguntará si deseamos instalar CLS ARM *Toolchain*, en caso de que no

esté instalado en nuestro ordenador. Haremos *click* en *Yes* para su instalación.

Como aclaración a este último paso, destacar que CSL ARM *Toolchain* contiene diversos elementos como por ejemplo el compilador GCCE, necesario para compilar aplicaciones para dispositivos reales de la serie S60.

7. Llegados a este punto ya dispondremos del emulador S60 y su SDK. Como último paso solo restará instalar en las aplicaciones de nuestro emulador móvil el intérprete de Python para S60, el cual nos permitirá ejecutar y probar *scripts* o programas desarrollados en este lenguaje para esta plataforma.

   Para ello descargaremos, de forma gratuita, el *plug-in* de Python para la versión de SDK Symbian anteriormente instalada, y de acuerdo con la serie del modelo de teléfono móvil para el que queramos desarrollar nuestras aplicaciones. En este sentido, nos dirigiremos a la página web del proyecto *SourceForge's* PyS60, y en nuestro caso concreto descargaremos el paquete `PythonForS60_1_4_5_3rdEdFP1.zip`. Para descargar Python for S60 1.4.5, 3ª Edición FP1 (*Feature Pack 1*) acudimos a la web: `http://sourceforge.net/projects/pys60`.

8. Una vez descargado, procedemos a descomprimir el paquete. Una vez realizada la descompresión observaremos dos archivos. Por un lado `dk_files.zip`, que contiene diversos ficheros estructurados de manera concreta, en diversas subcarpetas. Por otro lado encontraremos un *uninstaller script*, el cual no resulta relevante en la tarea de instalación. Deberemos mover los contenidos de `sdk_files.zip` al directorio `epoc32` de nuestro S60 SDK para C++ de nuestro ordenador respetando la estructura. En nuestro caso, dicho directorio será `"C:\Symbian\9.2\S60_3rd_fp1\Epoc32"`.

Una vez realizado el último paso, podremos lanzar nuestro emulador software desde `Todos los programas > S60 Developer Tools > 3rd Edition FP1 SDK > 1.0 > Emulator` (ver figura B.5).

Si hemos seguido correctamente los pasos anteriores, deberíamos tener un nuevo icono en nuestro menú de aplicaciones de nuestro emulador con el símbolo de Python, el cual nos dará acceso al intérprete y por tanto la ejecución de nuestros *scripts* (ver figuras B.6 y B.7).

Para comprobar que todo funciona correctamente podemos lanzar el intérprete de Python, desplazándonos previamente sobre él con las teclas de navegación del emulador y pulsando *Enter* después.

Figura B.5: Emulador S60 3rd Ed. FP1 SDK.

Figura B.6: Icono Python.

Figura B.7: Pantalla de bienvenida a Python.

Posteriormente podremos ejecutar algún *script* de ejemplo, ya incluido en la instalación. Para ello, podemos hacer *click* en `Options > Run Script: c:snake.py`. Debería aparecer en nuestra pantalla una aplicación clásica de los terminales Nokia del juego de la serpiente, la cual nos permita jugar durante unos breves segundos (ver figuras B.8 y B.9).

Existen multitud de ejemplos disponibles, a parte del anterior juego, para comprobar que la instalación ha sido exitosa y que todo funciona con normalidad.

Por último, conviene observar que los *scripts* Python de ejemplo se encuentran localizados en la carpeta `C:\Symbian\9.2\S60_3rd_FP1\epoc32\winscw\c\python`. Los nuevos programas que implementemos deberán estar ubicados en dicha carpeta, para poder ser ejecutados siguiendo el procedimiento del ejemplo anterior.

B.2.2. Instalación de la aplicación en el terminal móvil o emulador software

El usuario deberá almacenar, en la memoria `c` del terminal, los archivos pertenecientes a la aplicación móvil, o en la ruta `C:\Symbian\9.2\S60_3rd_FP1\epoc32\winscw\c\python`, en el caso de utilizar un emulador software. Dichos archivos son:

- `MS.py`
- `SIPmessages.py`
- `SDPmessages.py`
- `RTPpackets.py`
- `MJPEGpackets.py`
- `I2B.py`
- `ms_logo.jpeg`

Además, en el caso particular de utilizar un emulador software, desprovisto de cámara integrada, también será necesario el almacenamiento previo (en la misma ubicación) de las imágenes JPEG a enviar por la aplicación.

Figura B.8: Ejecución de un *script* en Python.

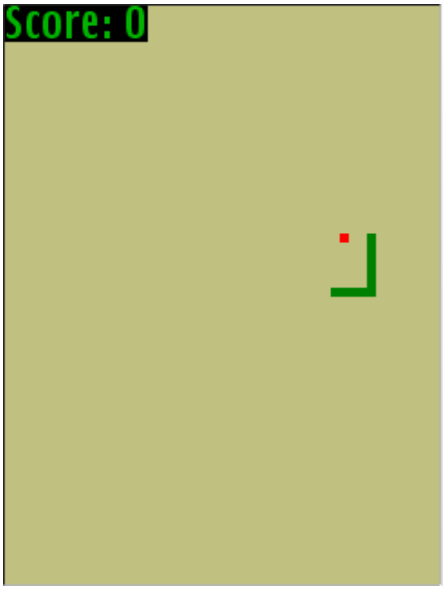

Figura B.9: Juego de la serpiente programado en Python.

Apéndice C

Manual de usuario

En este capítulo se explica como probar y ejecutar el sistema desarrollado durante el presente trabajo.

Para poner en marcha la aplicación práctica implementada deberemos seguir los siguientes pasos:

1. Lanzamos la aplicación receptora haciendo doble *click* sobre `CH.py`. Aparecerá la interfaz gráfica de usuario de la aplicación receptora (ver figura 5.4) y el intérprete de Python para PC, donde podremos visualizar los mensajes SIP/SDP intercambiados por el sistema (ver figura 5.5).

2. Configuraremos la IP y el puerto de la aplicación receptora donde se escucharán las comunicaciones SIP entrantes, provenientes de un terminal móvil. Para ello accedemos al menú *Configurar*.

3. Pulsamos el botón *Iniciar* de la interfaz gráfica de usuario de la aplicación receptora (ver figura 5.4).

4. Una vez que la aplicación receptora está inicializada correctamente, abriremos el intérprete de Python para S60, previamente instalado en el terminal o emulador (ver figura B.6), y ejecutaremos el *script* `MS.py` acudiendo a `Options > Run script`. Aparecerá la interfaz gráfica de usuario de la aplicación móvil (ver figura 4.6).

5. Configuramos la IP y el puerto de la aplicación móvil, las cuales deben coincidir con las anteriormente configuradas en la aplicación receptora. Para ello, pulsamos la tecla *Options* del terminal o emulador software y elegimos las diversas opciones (ver figura 4.7).

6. Pulsamos el botón *Conectar* de la interfaz gráfica de usuario de la aplicación móvil (ver figura 4.6). Se establecerá una sesión para el intercambio de datos multimedia y comenzará el envío de un flujo M-JPEG desde el terminal, así como su reproducción en el equipo receptor (ver figura 5.6).

7. En un momento determinado, pulsamos el botón *Cambiar de red* de la interfaz gráfica de usuario de la aplicación móvil (ver figura 4.6). Esta opción simulará un *hand-off* o cambio de red en el terminal, que será notificado al equipo reproductor, de tal forma que la reproducción del flujo M-JPEG enviado desde el móvil continuará de forma trasparente en el receptor. Este procedimiento puede simularse el número de veces que deseemos.

8. En un momento determinado, pulsamos el botón *Desconectar* de la interfaz gráfica de usuario de la aplicación móvil (ver figura 4.6). Esta opción cerrará la sesión multimedia entre el terminal móvil y el equipo receptor, finalizando el envío y reproducción del flujo M-JPEG.

Por último, para finalizar el sistema deberemos seguir los siguientes pasos:

1. Una vez el usuario pulsó el botón *Desconectar* de la interfaz gráfica de la aplicación móvil, puede finalizar dicha aplicación pulsando el botón *Exit* de su terminal o emulador software.

2. El usuario de la aplicación receptora pulsará el botón *Salir* de la interfaz gráfica de la aplicación reproductora.

Apéndice D

Manual de desarrollo de extensiones C++ para PyS60

D.1. Herramientas para construir una extensión C++ para PyS60

Para construir una extensión necesitaremos tener instaladas las siguientes herramientas en nuestro sistema operativo Microsoft Windows (XP/Vista/7):

1. S60 3ª Edición C++ SDK *Maintenance Release*, cuya instalación se detalla en la sección B del presente documento. Disponible en `http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html`.

2. Python for Symbian 60 SDK, cuya instalación se explica en la sección B del presente documento. Disponible en `http://sourceforge.net/projects/pys60/files/`.

3. Carbide C++ Express. Disponible en `http://www.forum.nokia.com/info/sw.nokia.com/id/dbb8841d-832c-43a6-be13-f78119a2b4cb.html`.

4. PyDev. Si decidimos usar Eclipse para desarrollar nuestras extensiones, instalaremos PyDev, un *plug-in* para este IDE que nos permite editar el código fuente Python. Disponible en `http://pydev.org/`.

5. Epydoc, una herramienta para generar la documentación del API (opcional). Disponible en `http://epydoc.sourceforge.net/index.html`

6. Subclipse, un *plug-in* para Eclipse que nos permite realizar un control de versiones (*Subversion*) (opcional). Disponible en `http://subclipse.tigris.org/`.

Se asume que si el lector ha llegado hasta el presente capítulo ya tiene correctamente instalado el SDK S60 3rd Edición C++, así como Python para dicho SDK para terminales Symbian S60. En caso contrario, se recomienda visitar el manual adjunto B en el presente documento para instalarlo.

Para comprobarlo, y siguiendo las instrucciones del Nokia Forum [21], abrimos la consola del sistema de Windows y tecleamos:

```
C:\>devices
```

Aparecerá por pantalla el SDK que tenemos instalado en nuestra máquina. En nuestro caso:

```
S60_3rd_FP1:com.nokia.s60 - default
```

En el caso de que tan solo nos aparezca `S60_3rd_FP1:com.nokia.s60` (sin *default*) deberemos configurar dicho SDK por defecto tecleando:

```
C:\>devices -setdefault @S60_3rd_FP1:com.nokia.s60
```

Este comando puede ser útil, si disponemos de varios SDK (por ejemplo FP1 y FP2) y queremos alternar entre ellos, fijando uno u otro por defecto.

Por otra parte, cuando indicábamos como instalar el SDK S60 3rd Edición, comentábamos que uno de los requisitos era tener instalado Perl (v5.6.1 o superior). Para comprobar que efectivamente tenemos dicha versión instalada, abriremos la consola del sistema y teclearemos:

```
C:\>perl -version
```

Análogamente, procederemos a comprobar que nuestra versión de Java es superior o igual a la 1.5 mediante:

```
C:\>java -version
```

D.2. Ejemplo de extensión C++ para PyS60: *uikludges.py*

En los siguientes epígrafes, vamos a describir como generar binarios y crear paquetes de instalación para PyS60 a partir de extensiones C++ para plataformas de 3ª Edición.

Como ya hemos comentado con anterioridad, es preciso disponer de los siguientes elementos:

- S60 3rd ed C++ SDK *maintenance release*.
- PyS60 1.3.8 (o superior) SDK.
- Opcionalmente, instalamos Carbide C++ *Express* como entorno de desarrollo para la extensión.

La mejor forma de entender este procedimiento es analizar un sencillo ejemplo, para posteriormente poder abordar nuestras propias extensiones.

Ya que el procedimiento de construir una extensión no es trivial, en este manual vamos a reproducirlo y documentarlo mediante un ejemplo sencillo, de dificultad similar al típico programa "Hola Mundo". Dicho ejemplo se encuentra disponible en el Wiki de Nokia [19] [20].

Concretamente, en este ejemplo vamos a utilizar un módulo llamado `uikludges.py` para mostrar como crear y construir una extensión. Dicho módulo `uikludges` nos ofrece la posibilidad de poder renombrar la etiqueta `Exit-key` en PyS60.

El objetivo de este ejemplo es conseguir renombrar dicha etiqueta, asignándole otro nombre (ver figura D.1). Como es lógico, este procedimiento no puede abordarse mediante los módulos estándar de PyS60, ya que dicha funcionalidad se encuentra en las "entraña" del código nativo C++ del dispositivo Symbian. Por esta razón, generaremos una extensión C++ para Python, que nos permita construir una sencilla función PyS60, desde la cual poder realizar dicha tarea con suma facilidad. Por ejemplo, cambiaremos la palabra "Exit" por "Back".

La primera tarea que tenemos que realizar es comprobar el código fuente de la extensión de ejemplo. En este paso, resulta interesante tener instalado un cliente de *subversion* (control de versiones) si no lo tenemos hecho ya. En este ámbito, uno de los clientes de subversión más populares es SVN.

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias.

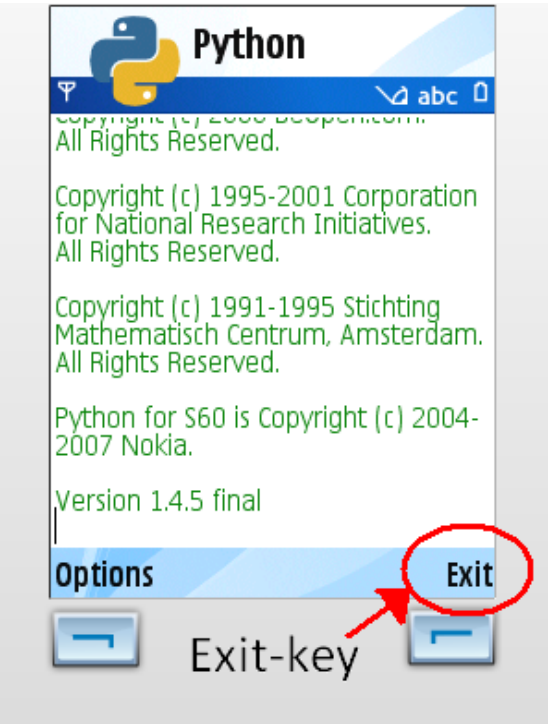

Figura D.1: Creando extensión para renombrar la etiqueta “Exit-key”.

Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como *svn*, por ser ese el nombre de la herramienta de línea de comandos.

Una característica importante de *Subversion* es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Entre las diferentes ventajas derivadas de utilizar este tipo de herramientas podemos destaca el hecho de que podemos seguir la historia de los archivos y directorios a través de copias y renombrados, así como las modificaciones (incluyendo cambios a varios archivos) atómicas.

Podemos descargarnos *svn* desde su sitio web oficial en `http://subversion.tigris.org/`.

Posteriormente abriremos la línea de comandos y teclearemos:

```
% cd <your_work_folder>
% svn export https://pymbian.svn.sourceforge.net/svnroot/
                        pymbian/uikludges/trunk/_uikludges
```

En la anterior URL disponemos de todo el código fuente que necesitaremos, así como otros archivos, para generar nuestra extensión. Recordemos que el objetivo de este ejemplo es utilizar un módulo denominado `uikludges` para ilustrar como crear y generar una extensión C++ para PyS60.

Los archivos de este ejemplo son los siguientes:

- `bld.inf`: Define los componentes (archivos .mmp) y las plataformas objetivo (Windows o dispositivo) para las cuales construiremos la extensión.
- `uikludges.mmp`: Contiene los archivos con el código fuente C++, UID, requeridos por las *capabilities*.
- `uikludges.rss`: Definiciones Fuente para el proyecto.
- `uikludges.cpp`: Código fuente C++ para generar la asociaciones con el código nativo para nuestra extensión (será compilado a archivo `DLL_uikludges.pyd`). En otras palabras, contiene nuestra extensión en código C++.

- `uikludges.py`: Módulo Python que nos permite manejar la extensión que vamos a generar.
- `uikludges.pkg`: Contiene instrucciones para generar los archivos SIS, los cuales nos servirán para instalar la extensión en nuestro teléfono.
- `_build`: Esta es la carpeta destino donde irán a parar los archivos SIS generados.
- `uikludges.sis`: Es el archive Symbian listo para distribuir y que contiene la aplicación.
- `Makefile`: Contiene las típicas instrucciones para realizar tareas comunes, como la generación de documentación de la API, ago que no está directamente disponible en el IDE de Carbide.

De todos estos archivos, hay dos fundamentales dentro del paquete de ficheros del presente ejemplo:

- `uikludges.cpp`: Contiene la extensión C++ propiamente dicha. Disponible en `http://sourceforge.net/apps/trac/sourceforge/wiki/Subversion`.
- `uikludges.py`: Para 3ª Edición se requiere de una *extension loader*. Disponible en `http://sourceforge.net/apps/trac/sourceforge/wiki/Subversion`.

El resto de los archivos son sólo material para la construcción y la instalación, por lo que no deben ser examinados a fondo por el lector.

Algo importante a tener en cuenta es la estructura de directorios de nuestro ejemplo, la cual se muestra en la figura D.2:

El directorio `keys` contiene las claves para el firmado de nuestra aplicación, algo totalmente necesario como ya explicamos en epígrafes anteriores. La carpeta `python` contiene el código generado por nosotros, que será utilizado para invocar nuestra extensión C++ desde nuestro programa Python para S60. El directorio `src` contiene los ficheros fuente con el código C++ de nuestra extensión, y representa la carpeta fundamental del ejemplo, ya que contiene la funcionalidad de la extensión en código Symbian C++ nativo.

Si queremos abrir nuestro proyecto con Carbide, debemos ubicar esta estructura de carpetas en nuestro *workspace*.

Por ejemplo, en nuestro caso estaría ubicada en:

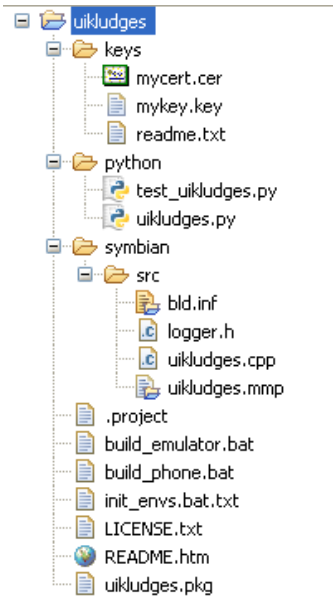

Figura D.2: Estructura de directorios de uikludges para renombrar "Exit-key".

`C:\Symbian\Carbide\workspace\uikludges`

Para abrir el proyecto con Carbide, seguiremos los pasos que se relatan a continuación.

Abrimos Carbide y nos dirigimos a `File > New > Project...` Se nos abrirá una ventana donde seleccionaremos `General > Project` y haremos clic en `Next`.

En la siguiente ventana elegiremos un nombre para nuestro ejemplo. En nuestro caso `uikludges`, ya que deseamos que Carbide detecte la estructura de directorios y los archivos de nuestro pequeño proyecto. Hacemos clic en `Next` y `Finish`. Carbide cargará nuestro proyecto para poder empezar a trabajar con él.

Hechas todas estas matizaciones, vamos a comenzar el proceso de crear nuestra extensión. Este ejemplo incluye dos ficheros `.bat` que automatizan el proceso de crear nuestra extensión C++ para PyS60: `build_emulator.bat` (que automatiza la construcción de la extensión para que funcione en nuestro emulador para S60) y `build_phone.bat` (que automatiza la construcción de la extensión para que funcione en nuestro terminal móvil).

En cualquier caso, en esta breve guía vamos a estudiar como generar la extensión directamente usando la línea de comandos de Windows. Carbide realiza estas mismas operaciones a través de su interfaz gráfico, por lo que el resultado

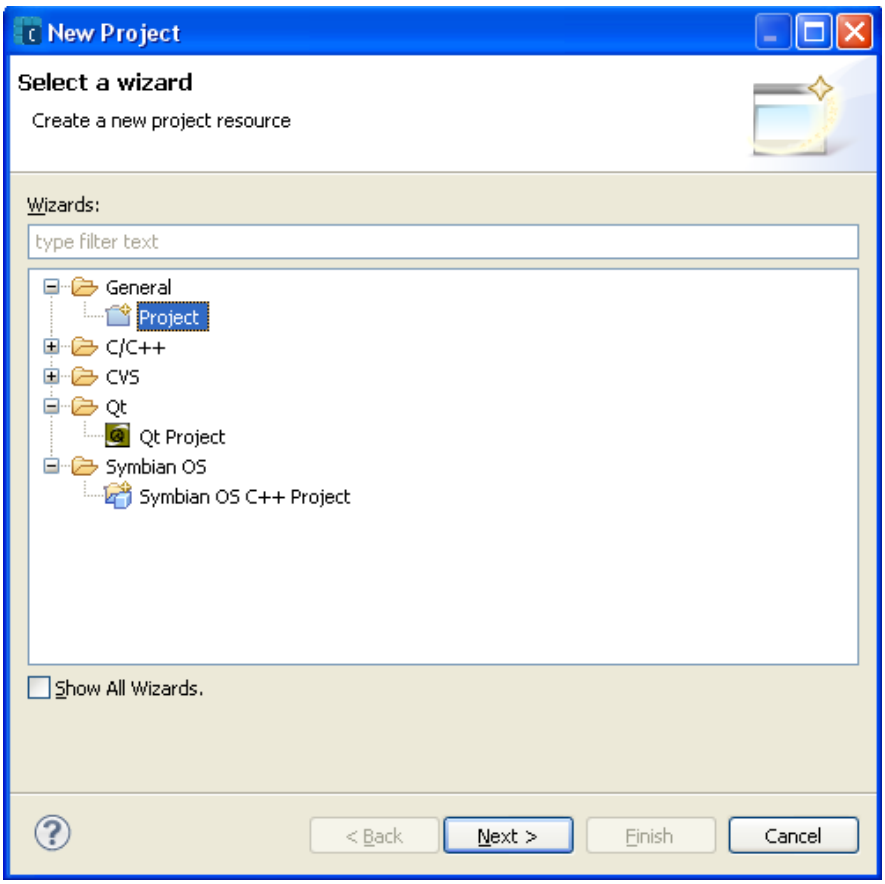

Figura D.3: Creando un nuevo proyecto en Carbide.

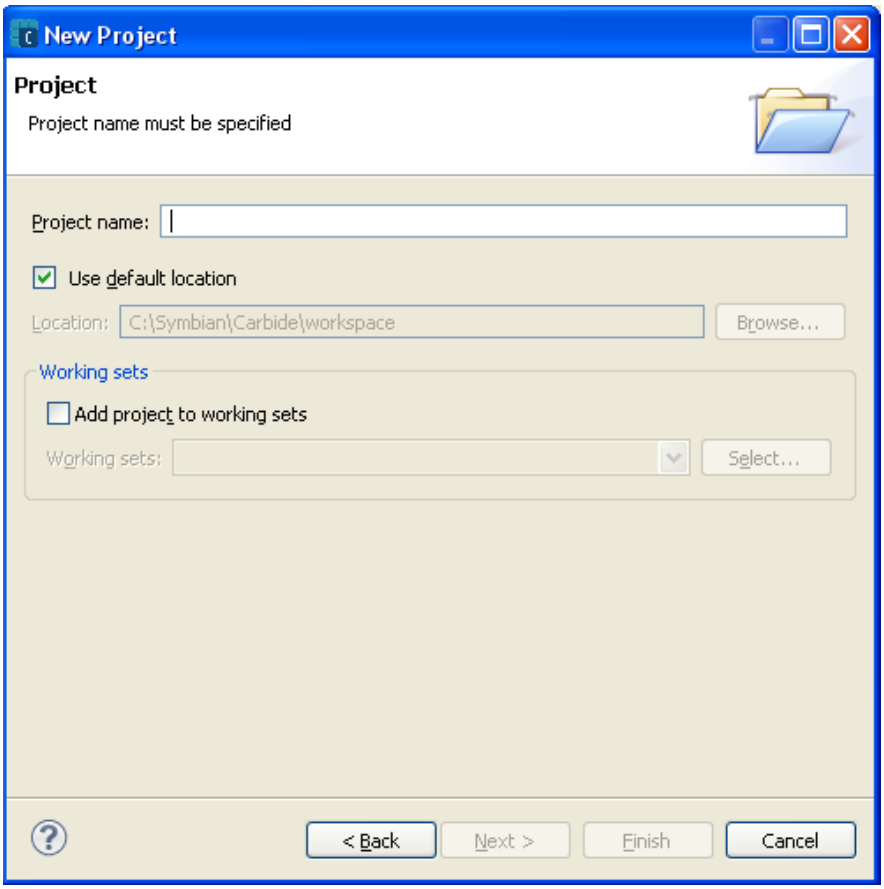

Figura D.4: Dando un nombre al proyecto en Carbide.

debe ser exactamente el mismo en ambos casos.

D.3. Generando la extensión para nuestro emulador software para S60

Primero, abrimos la consola y nos situamos en el directorio donde tengamos el código fuente. Los comandos que debemos introducir son los siguientes:

```
pushd .
cd symbian\src\
bldmake bldfiles
call abld reallyclean winscw udeb
call abld build winscw udeb
popd
copy python\uikludges.py %SDKFOLDER%\Epoc32\winscw\c\resource
copy python\test_uikludges.py %SDKFOLDER%\Epoc32\winscw\c\python
copy %SDKFOLDER%\Epoc32\release\winscw\udeb\_uikludges.pyd
             %SDKFOLDER%\Epoc32\winscw\c\sys\bin\_uikludges.pyd
```

`pushd` es un comando para plataformas DOS/Windows que nos sirve para situarnos en un directorio concreto y almacenar el directorio actual en una pila FILO (`popd` realiza la tarea contraria).

Como se puede observar, lo primero que hacemos es almacenar el directorio actual. Seguidamente accedemos a los ficheros fuente y los compilamos, construyendo posteriormente la extensión (limpiando previamente el contenido que pudiera haber en `udeb`).

Por último se copian los archivos a las carpetas correspondientes. Este proceso también se puede realizar manualmente arrastrando los ficheros mediante el explorador de carpetas de Windows.

Si abrimos nuestro emulador, y ejecutamos el *script* Python `test_uikludges.py`, el resultado debería ser el mostrado en la figura D.5(donde podemos apreciar que hemos cambiado el nombre del título de la tecla de navegación, gracias a nuestra extensión).

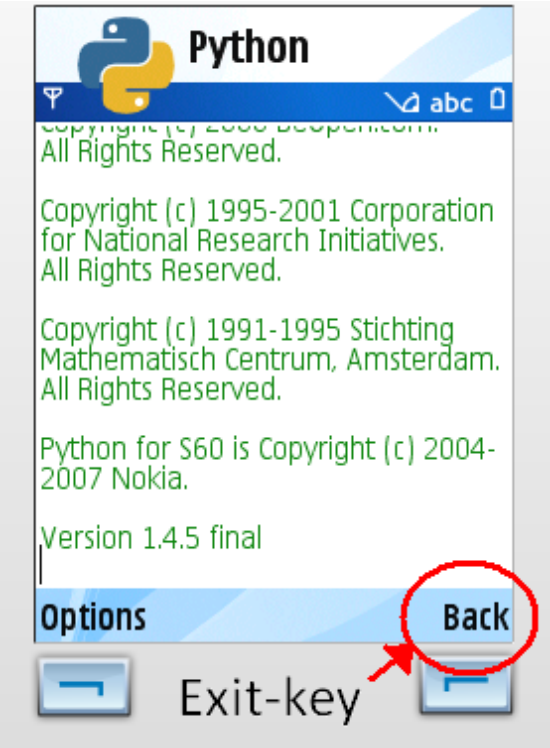

Figura D.5: Etiqueta “Exit-key” renombrada como “Back”.

D.4. Generando la extensión para nuestro teléfono

Abrimos la línea de comandos de Windows y nos situamos en el directorio donde tenemos el código fuente (ver figura D.6):

```
% cd symbian\src
```

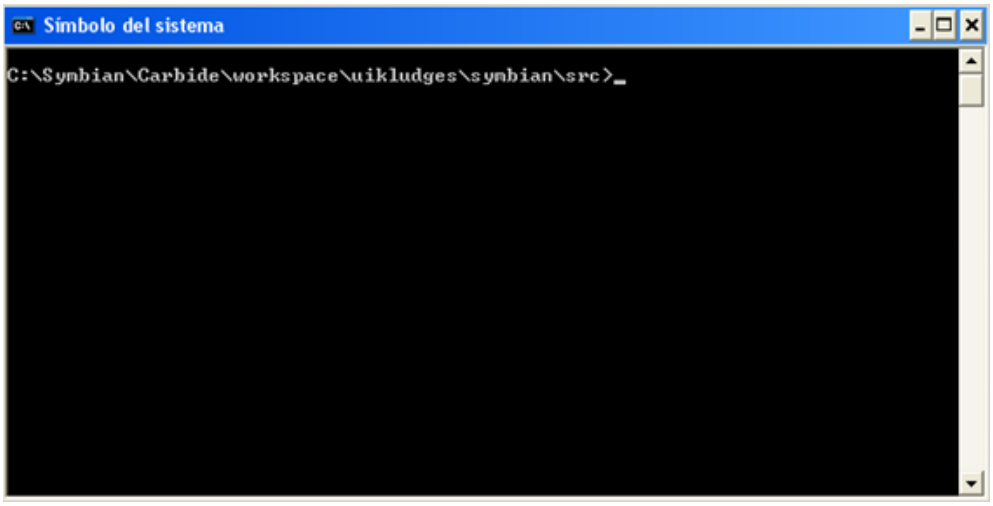

Figura D.6: Abrimos la línea de comandos en el directorio donde tenemos el código fuente.

Compilamos nuestros archivos (ver figura D.7):

```
% bldmake bldfiles
```

Construimos nuestra extensión (ver figura D.8):

```
% abld build gcce urel
```

Llegados a este punto deberíamos tener un fichero binario en el directorio donde tengamos instalado nuestro SDK:

```
<SDK path>\epoc32\release\gcce\urel\_uikludges.pyd
```

En este caso concreto está ubicado en:

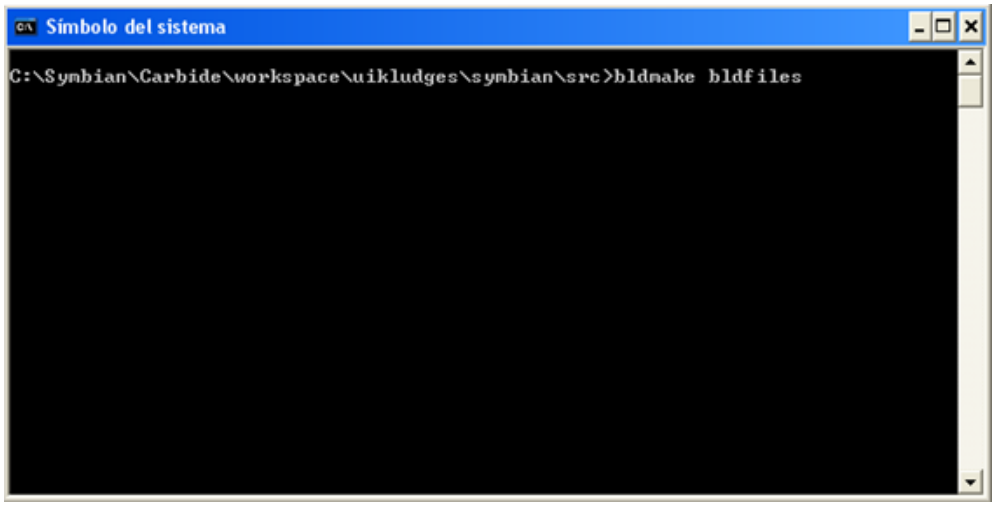

Figura D.7: Compilamos nuestros archivos.

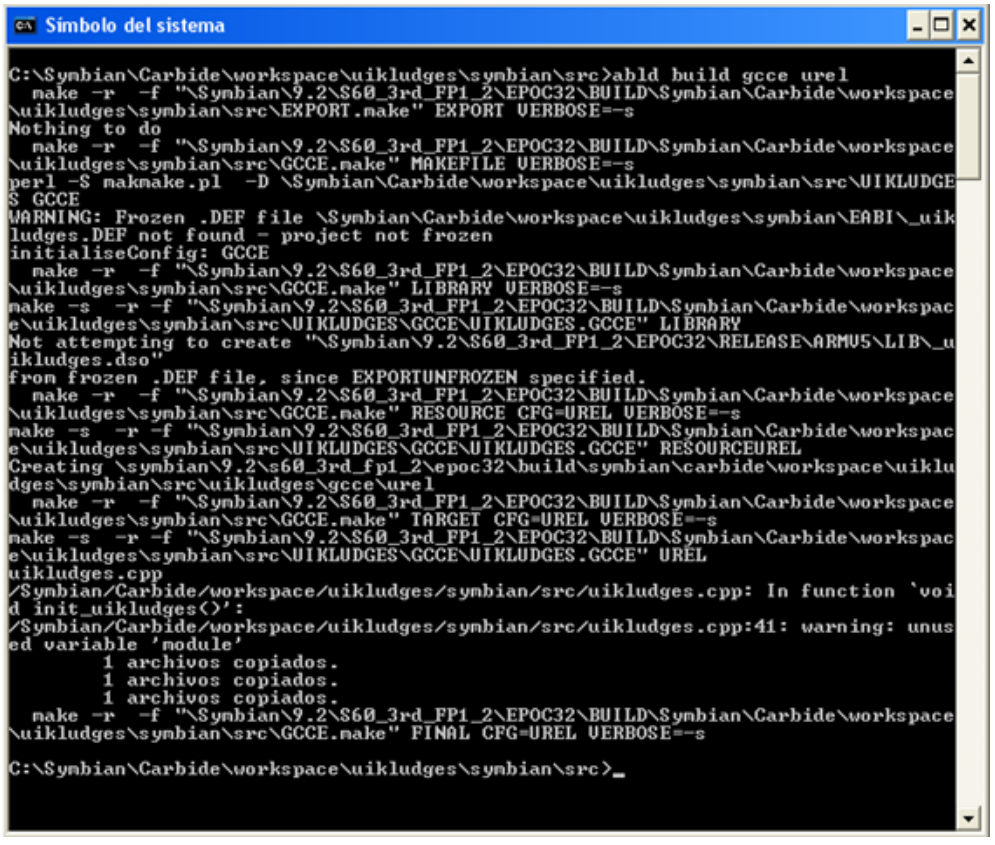

Figura D.8: Construimos nuestra extensión.

```
C:\Symbian\9.2\S60_3rd_FP1\Epoc32\release\gcce\urel\_uikludges.pyd
```

Cabe destacar que si no tenemos instalado WINS (*Windows Emulator*) *tool chain*, necesitaremos borrar `wins` de las secciones `PRJ_Plataforms` en el fichero `bld.inf` u obtendremos un error. En nuestro caso concreto, esto no ha sucedido ya que disponemos de WINS *tool chain* en nuestro equipo.

D.5. Generando un paquete de instalación

En el punto anterior hemos conseguido obtener la extensión para nuestro teléfono, pero en la práctica, las aplicaciones distribuirse en un paquete de instalación. Por esta razón, en este paso vamos a obtener un fichero `uikludges.sis`, que ya está en el típico formato en el que se suelen proporcionar los archivos para terminales móviles.

Si echamos un vistazo a `uikludges.pkg` podemos darnos de cuenta de que este archivo define instrucciones relacionadas con la instalación de los ficheros.

Para obtener `uikludges.sis` tenemos que ejecutar:

```
% mkdir _build
% copy <SDK path>\epoc32\release\gcce\urel\_uikludges.pyd _build\
% makesis uikludges.pkg
```

D.6. Firmando nuestro paquete

Por cuestiones de seguridad, las aplicaciones para Symbian deben estar firmadas por el desarrollador o empresa que las ha desarrollado. Firmar un paquete consiste simplemente en aplicarle un *password*, es decir, un simple número, el cual está relacionado con la empresa que ha implementado dicho software. Puesto que las firmas son asignadas por Nokia, y requieren de un proceso de solicitud y registro, se garantiza y certifica que las aplicaciones que instalamos en el teléfono tienen un comportamiento adecuado, aunque hayan sido desarrolladas por un tercero (si no existiera este mecanismo, un atacante podría crear un software malicioso que recopilara información sobre el usuario, se enviara sus mensajes, o cualquier otro comportamiento malévolo que se nos ocurra).

De este modo, para que podamos enviar nuestro paquete de instalación al teléfono, es necesario que esté firmado con anterioridad.

Concretamente, en el ejemplo que estamos tratando utilizaremos el método de *self-singning* o autofirmado, por resultar el más simple de todos.

Para poder autofirmar nuestro paquete, primero necesitamos generar un certificado y una clave privada con la orden:

```
% makekeys -cert -password yourpassword -len 2048 -dname
          "CN=uikludges OU=Development" mykey.key mycert.cer
```

Tras introducir este comando, necesitaremos presionar el teclado para generar una secuencia aleatoria.

El siguiente paso consistirá en utilizar la clave obtenida anteriormente para firmar el paquete mediante la orden:

```
% signsis uikludges.sis uikludges.sisx mycert.cer mykey.key
                                                    yourpassword
```

Es importante destacar que el `.pkg` utiliza el UID, el cual es usado para identificar la instalación del paquete. El lector puede obtener sus propios UIDs registrándose en `http://symbiansigned.com` y solicitando los UDIs.

Si el lector desea ampliar su conocimiento sobre firmado de paquetes, le recomiendo visitar Symbian signed [22] y el Nokia Forum [3].

Apéndice E

Glosario de términos

Atributo Es aquella información asociada a una entidad que especifica una característica de la misma y que permite describirla. En un programa Python, cualidades de un objeto y/o clase.

BTS Del inglés *Base Transceiver Station*, equipo que facilita la comunicación inalámbrica entre un equipo de usuario o terminal móvil, y la red.

CH Del inglés *Corresponding Host*, máquina o equipo correspondiente. En el contexto de este documento, el equipo receptor.

Datagrama Técnica por la que cada paquete se trata de forma independiente, conteniendo cada uno la dirección de destino.

Entidad Es alguien o algo (un sistema *software*, un objeto, un dispositivo, una persona, etc.) que se caracteriza a través de la medida de sus atributos.

FLOSS *Free and Open Source Software*, software libre y de código abierto.

GNU Acrónimo recursivo del inglés *GNU is Not Unix*, es un proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre, el sistema GNU, apelando a los valores y el espíritu de cooperación entre usuarios de computadoras.

GPL Es la Licencia Pública General de GNU o más conocida por su nombre en inglés *GNU General Public License*, o simplemente sus siglas del inglés GNU GPL. Fue creada por la *Free Software Foundation* en 1989, y está orientada a proteger la libre distribución, modificación y uso de software. Su propósito es proteger al software libre de intentos de apropiación que restrinjan esas libertades a los usuarios.

Hand-off Cambio de red o dominio.

HTTP Del inglés *HyperText Transfer Protocol*, es un protocolo de transferencia de hipertexto, orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

JPEG Del inglés *Joint Photographic Experts Group* (nombre de la comisión que creó la norma JPEG), es un método de compresión de imágenes a menudo considerado como un formato de archivo. JPEG/Exif es el formato de imagen más común utilizado por las cámaras fotográficas digitales y otros dispositivos de captura de imagen, junto con JPEG/JFIF, que también es otro formato para el almacenamiento y la transmisión de imágenes fotográficas en Internet.

MIME Del inglés *Multipurpose Internet Mail Extensions*, es una serie de convenciones o especificaciones dirigidas al intercambio, a través de Internet, de todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario.

MS Del inglés *Mobile Station*, teléfono, dispositivo o estación móvil.

M-JPEG *Motion JPEG*, es un formato multimedia donde cada fotograma o campo entrelazado de una secuencia de vídeo digital es comprimida por separado como una imagen JPEG. Es usado con frecuencia en dispositivos portátiles tales como cámaras digitales.

PIL Del inglés *Python Imaging Library* es una librería externa para el lenguaje de programación Python que añade soporte para la apertura, manipulación y guardado de imágenes en diferentes formatos, soportando versiones Python desde la 2.2 a la 2.6.

PyS60 Abreviatura de "Python para S60", constituye un conjunto de módulos o librerías Python basadas en extensiones Symbian C++ y enfocadas a la creación de aplicaciones para terminales móviles Symbian S60.

Python Python es un lenguaje de programación creado por Guido van Rossum a principio de los años 90. Es un lenguaje interpretado o de *script*, con tipado dinámico, fuertemente tipado, orientado a objetos y multiplataforma. En algunos contextos, puede afirmarse que Python guarda un cierto parecido a Perl, y su nombre está inspirado en el grupo de cómicos ingleses "Monty Python".

SDP Del inglés *Session Description Protocol*, es un protocolo para describir los parámetros de inicialización de los flujos multimedia intercambiados en una sesión.

Software libre Aquel software protegido bajo la licencia GPL.

Symbian Es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran: Nokia, Sony Ericsson, Psion, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus orígenes provienen de su antepasado EPOC32, utilizado en PDA's y *Handhelds* de PSION.

TCP Del inglés *Transmission Control Protocol*, es uno de los protocolos fundamentales en Internet, mediante el cual, multitud de programas de una computadora pueden establecer conexiones con otras aplicaciones pertenecientes a una red para enviarse flujos de datos.

UDP Del inglés *User Datagram Protocol*, es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera.

URL Del inglés *Uniform Resource Locator*, es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, videos, presentaciones digitales, etc.

VLC *"Video LAN Client"*, es un reproductor y *framework* multimedia del proyecto VideoLAN; proyecto de software libre distribuido bajo la licencia GPL.

Bibliografía

[1] Varios autores. S60 Platform SDKs for Symbian OS, for C++, Nokia Forum. 2010. Disponible en http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html

[2] Symbian.org. 2010. Disponible en http://www.symbian.org/

[3] Documentación oficial del Foro de Nokia, Nokia Forum. 2010. Disponible en http://www.forum.nokia.com/

[4] Dispositivos Symbian. 2010. Disponible en http://www.symbian.org/devices

[5] Lenguaje de programación Python. Python Software Foundation. Disponible en http://www.python.org

[6] Documentación oficial del lenguaje de programación Python. Disponible en http://www.python.org/doc/

[7] Varios autores. Python for S60, Nokia Corporation. 2009. Disponible en http://wiki.opensource.nokia.com/projects/Python_for_S60

[8] Varios autores. PyS60 Library Reference, Nokia Corporation. 2008.

[9] Documentación oficial de la librería Python TkInter. Disponible en http://wiki.python.org/moin/TkInter

[10] Documentación oficial de la librería PIL. Disponible en http://www.pythonware.com/products/pil/

[11] Varios autores. Installing PyS60, Nokia Corporation. 2009. Disponible en http://wiki.opensource.nokia.com/projects/Installing_PyS60

[12] Marc-André Lemburg. DB-API 2.0 interface for SQLite databases, The Python Standard Library, Python Software Foundation. 2010. Disponible en http://docs.python.org/library/sqlite3.html

[13] Marc-André Lemburg. Python Database API Specification v2.0, PEP 249, Python Software Foundation. Marzo 2008. Disponible en http://www.python.org/dev/peps/pep-0249/

[14] R. González Duque. Python para todos. 2007. Disponible en http://mundogeek.net/tutorial-python/

[15] R. González Duque. Guía de estilo del código Python. 2007. Disponible en http://mundogeek.net/traducciones/guia-estilo-python.htm

[16] Anand Singh, Ashwin Rao, Jukka Laurila, Rajatah Kamath, Mahesh S., T.S. Vijayan. Documentación oficial de la librería de módulos PyS60 para terminales Symbian S60. Disponible en http://sourceforge.net/projects/pys60/

[17] Frank H.P. Fitzek, F. Reichert. Mobile Phone Programming and its applications to wireless networking, Ed. Springer. 2007.

[18] Michael A. Cvington. Getting Started with Python in IDLE, Artificial Intelligence Center Universidad de Georgia. 2004. Disponible en http://www.ai.uga.edu/mc/idle/index.html

[19] Varios autores. PyS60 creating extensions, Wiki de Nokia. Abril, 2009. Disponible en http://wiki.opensource.nokia.com/projects/PyS60_creating_extensions

[20] Varios autores. PyS60 extensions creation using Carbide, for C++, Wiki de Nokia. Octubre, 2008. Disponible en http://wiki.opensource.nokia.com/projects/PyS60_extensions_creation_using_Carbide

[21] Varios autores. How do I start programming for Symbian OS?, Nokia Forum. 2010. Disponible en http://wiki.forum.nokia.com/index.php/Getting_started

[22] Varios autores. Symbian signed, Symbian Foundation Ltd. 2009. Disponible en http://symbiansigned.com

[23] F. Vakil, A. Dutta, M. Tauil, S. Baba, N. Nakajima, Y. Shobatake, H. Schulzrinne. Supporting Mobility for Multimedia with SIP, Internet Draft, IETF Network Working Group. Julio, 2001. Disponible en http://www.argreenhouse.com/SIP-mobile/sip_draft1

[24] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. SIP: Session Initiation Protocol, RFC 3261, IETF Network Working Group. Junio, 2002. Disponible en `http://www.ietf.org/rfc/rfc3261.txt`

[25] J. Rosenberg. The Session Initiation Protocol (SIP) UPDATE Method, RFC 3311, IETF Network Working Group. Septiembre, 2002. Disponible en `http://www.ietf.org/rfc/rfc3311.txt`

[26] M. Handley, V. Jacobson, C. Perkins. SDP: Session Description Protocol, RFC 4566, IETF Network Working Group. Julio, 2006. Disponible en `http://www.ietf.org/rfc/rfc4566.txt`

[27] H. Schulzrinne, S Casner. RTP Profile for Audio and Video Conferences with Minimal Control, RFC 3551, IETF Network Working Group. Julio, 2003. Disponible en `http://www.ietf.org/rfc/rfc3551.txt`

[28] L. Berc, W. Fenner, R. Frederick, S. McCanne, P. Stewart. RTP Payload Format for JPEG-compressed Video, RFC 2435, IETF Network Working Group. Octubre, 1998. Disponible en `http://www.ietf.org/rfc/rfc2435.txt`

[29] H. Hoffman, G. Fernando, V. Goyal, M. Civanlar. RTP Payload Format for MPEG1/MPEG2 Video, RFC 2250, IETF Network Working Group. Enero, 1998. Disponible en `http://tools.ietf.org/html/rfc2250`

[30] David H. Crocker. Standard for the format of ARPA Internet text messages, RFC 822, Dept. of Electrical Engineering University of Delaware. Agosto, 1982. Disponible en `http://www.ietf.org/rfc/rfc822.txt`

(Fecha de los documentos consultados en Internet [1])

[1] Todas las fuentes consultadas en Internet han sido comprobadas a día 27 de octubre de 2010